НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Програмне забезпечення інформаційно-комунікаційних систем»
спеціальності 121 «Інженерія програмного забезпечення»
на тему: «Система віддаленого моніторингу засобів комунального обліку»

Виконав (-ла):
студент (-ка) IV курсу, групи ІТ-61
Тимченко Олексій Юрійович _____

Керівник:
Старший викладач каф. АУТС
Тимофєєва Юлія Сергіївна _____

Рецензент:
Доцент каф. ОТ
Волокита Артем Миколайович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Програмне забезпечення інформаційно-комунікаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студенту

Тимченку Олексію Юрійовичу

1. Тема проєкту «Система віддаленого моніторингу засобів комунального обліку», керівник проєкту Тимофєєва Юлія Сергіївна, старший викладач кафедри АУТС, затверджені наказом по університету від «07» травня 2020 р. № 1081-с.

2. Термін подання студентом проєкту: 9 червня 2020 р.

3. Вихідні дані до проєкту: розподілені системи, ASP.NET Core, мікроконтролер ESP32, платформа Microsoft Azure, мова програмування C#.

4. Зміст пояснювальної записки: аналіз вимог та існуючих рішень, технології і підходи використані при розробці, програмна реалізація, розгортання і супровід системи.

5. Перелік графічного матеріалу: схема бази даних, діаграма класів проєкту DataAccess, діаграма класів проєкту MonitoringSpa , діаграма розгортання

6. Дата видачі завдання: 5 березня 2020 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Аналіз джерел за тематикою проєкту	10.04.2020	
2.	Аналіз існуючих рішень	17.04.2020	
3.	Розроблення функціональної схеми системи	24.04.2020	
4.	Розроблення структурної схеми для мікроконтролерної підсистеми	08.05.2020	
5.	Виготовлення макету та програми для мікроконтролерної підсистеми	08.05.2020	
6.	Розроблення backend частини системи	13.05.2020	
7.	Розроблення додатку для представ-лення даних	13.05.2020	
8.	Налаштування та тестування проду-кту	15.05.2020	
9.	Розробка графічних матеріалів	20.05.2020	
10.	Розробка пояснювальної записки	24.05.2020	
11.	Подання проєкту на основний захист	15.06.2020	

Студент Олексій ТИМЧЕНКО

Керівник Юлія ТИМОФЄЄВА

АНОТАЦІЯ

Тимченко О.Ю. Система віддаленого моніторингу засобів комунального обліку. КПІ ім. Ігоря Сікорського Київ, 2020.

Проєкт складається з 4-х розділів, містить 60 сторінок, 27 рисунків, 1 таблицю, посилання на 17 джерел та 4 кресленики.

Об’єктом дослідження є системи для віддаленого моніторингу засобів комунального обліку.

Метою дипломного проєкту є спрощення та полегшення процесу моніторингу засобів комунального обліку.

Перший розділ містить аналіз існуючих на ринку аналогічних рішень та формулювання вимог до системи моніторингу.

У другому розділі розглядаються технології та принципи, що використані для розробки, обґрунтовується їх вибір.

Третій розділ містить опис та пояснення функціонування програмної реалізації побудованої системи.

В четвертому розділі було наведено опис графічного інтерфейсу системи та інструкцію для розгортання системи.

Отримані результати можуть бути використані для подальшої розробки та розширення функціональності системи, що була розроблена або для побудови аналогічних систем.

Ключові слова: Microsoft Azure; система моніторингу; веб-застосунок; ASP.NET Core; Angular; мікроконтролерна система.

ABSTRACT

Tymchenko O.Y. Remote utility monitoring system. Igor Sikorsky KPI, Kyiv, 2020.

The project consists of 4 sections, contains 60 pages, 27 figures, 1 table, 17 references to sources and 4 drawings.

The object of research is systems for remote monitoring of utility

The aim of the project is to automate the process of utility monitoring.

First section contains an analysis of similar market solutions and development of requirements for the monitoring system.

Second section is concentrated on technologies and principles that were used in the process of development.

Third section provides description and explanation of the implementation of the created software.

Fourth section contains user instructions and instructions for system deployment.

Obtained results can be used for further improvement and development of the created system or in the process of new system creation.

Keywords: Microsoft Azure; monitoring system; web application; ASP.NET Core; Angular; microcontroller system.

Поз.	Формат	Позначення	Найменування	Кількість аркушів	№ прим.	Примітки
1			Документація загальна			
2						
3			Знову зроблена			
4						
5	А4	ІТ61.250БАК.004 ПЗ	Система віддаленого моніторингу	60		
6			засобів комунального обліку.			
7			Пояснювальна записка			
8	А3	ІТ61.250БАК.004 Д1	Система віддаленого моніторингу	1		
9			засобів комунального обліку.			
10			Схема бази даних			
11	А3	ІТ61.250БАК.004 Д2	Система віддаленого моніторингу	1		
12			засобів комунального обліку.			
13			Діаграма класів проєкту			
14			DataAccess			
15	А3	ІТ61.250БАК.004 Д3	Система віддаленого моніторингу	1		
16			засобів комунального обліку .			
17			Діаграма класів проєкту			
18			MonitoringSpa			
19	А3	ІТ61.250БАК.004 Д4	Система віддаленого моніторингу	1		
20			засобів комунального обліку.			
21			Діаграма розгортання			
22						
23						
24						

					ІТ61.240БАК.004 ТП		
Змн.	Лист	№ докум.	Підпис	Дата			
Розроб.		Тимченко О.Ю.			Система віддаленого моніторингу засобів комунального обліку. Відомість технічного проєкту	Літ.	Лист 1 / Листів 1
Перевір.		Тимофєєва Ю.С					
Н. Контр.						КПІ ім. Ігоря Сікорського кафедра АУТС гр. ІТ-61	
Затверд.							

Пояснювальна записка
до дипломного проєкту
на тему: «Система віддаленого моніторингу засобів комунального обліку»

Київ – 2020 рік

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

БД — база даних.

СУБД — система управління базами даних.

API — application programming interface. Прикладний програмний інтерфейс.

Backend — програмно-апаратна частина система.

COM порт — communications port. Послідовний порт для зв’язку між компонентами системи.

CRUD — create read update delete. Створення, читання, редактування та видалення.

CSS — cascading style sheets. Мова для описання зовнішнього вигляду документу.

DI — dependency injection. Впровадження залежностей.

EF — entity framework. ORM для роботи на платформі .NET.

Frontend — клієнтська частина користувацького інтерфейсу системи.

GPIO — general purpose input output. Інтерфейс для зв’язку між компонентами системи.

HTML — hypertext markup language. Стандартна мова розмітки веб-документів.

HTTP — hyper text transfer protocol. Протокол програмного рівня для передачі даних для розподілених систем.

IDE — integrated development environment. Програмне забезпечення, що призначене для зручної розробки програмного забезпечення.

JWT — JSON web token. Відкритий стандарт для опису шляху для захищеного обміну інформацією у форматі JSON.

LINQ — language-integrated query. Набір технологій, що дають можливість писати запити до джерел даних мовою C#.

MPA — multiple page application. Веб-застосунок, який працює на сервері і потребує перезавантаження сторінок.

MVC — model view controller. Модель, вигляд, контролер.

MVP — minimal valuable product. Продукт з мінімальною кількістю функціоналу, яким можна користуватись.

ORM — object-relational mapping. Об’єктно-реляційна проекція.

PDF — portable document format. Формат портативних документів.

RBAC — role-based access control. Керування доступом на основі ролей.

REST — representational state transfer. Підхід до архітектури взаємодії між компонентами в розподіленій системі.

SPA — single page application. Веб-застосунок який працює в браузері і не потребує перезавантаження сторінок.

SQL — structured query language. Декларативна мова для управління даними в реляційній базі даних.

ULP — ultra-low power. Позначення для пристроїв, які можуть споживати дуже малу кількість енергії.

UML — unified modeling language. Мова графічного опису для об’єктного моделювання в розробці програмного забезпечення.

Wi-Fi — технологія бездротової передачі даних.

ВСТУП

У сучасному світі, темп життя підвищується, і людська увага стає одним із найважливіших ресурсів. Стрес як явище, негативно впливає на здатність концентруватись на дійсно важливих задачах, які позитивно змінюють життя. Важливим аспектом зниження суттєвої кількості стресу є зменшення кількості рутинних справ шляхом їхньої автоматизації. Система що була розроблена, покликана автоматизувати одну з таких задач — моніторинг засобів комунального обліку.

У процесі обслуговування певної нерухомості, невідворотно доводиться стикатися з необхідністю перевірки показань та стану встановлених засобів обліку. Зазвичай, такі засоби встановлені у не дуже зручних та доступних місцях, тому кожен акт перевірки є доволі трудомістким та тривалим, ба більше, такі операції є періодичними, тому необхідно пам’ятати про необхідність виконання цієї операції. Також, при необхідності проаналізувати рівень споживання того чи іншого ресурсу немає способу зробити це просто та інтуїтивно, доводиться збирати платіжні квитанції, або вести облік самостійно в електронному або паперовому виді. Такі рутинні задачі мають бути автоматизовані шляхом використання сучасних технологій, що в цілому покращить якість людського життя. Очевидно, що перекладання частини таких зобов’язань на програмний продукт має дуже велику кількість переваг, наприклад, можливість перевіряти поточні показання засобів обліку у будь-якому куточку світу. В подальшому, система може бути розширена та інтегрована з сервісами для оплати комунальних послуг та інтегрована в систему розумного будинку як одна із його складових. Також одним із можливих покращень може бути конфігурація пристрою для збору даних за допомогою веб-сторінки або мобільного додатку.

Розробка дипломного проєкту має на меті спрощення процесу моніторингу засобів комунального обліку шляхом створення віддаленої, розподіленої, масштабованої та безкоштовної системи, яка зможе містити у собі функціональ-

ність для моніторингу засобів обліку та подальшого збереження отриманих даних, їх відображення у текстовому та графічному вигляді. Задачею є аналіз існуючих варіацій таких систем на ринку, розробка архітектури та програмного коду для системи моніторингу та налагодження її безперервної та безвідмовної роботи. Для цього було використано ряд технологій, таких як новий фреймворк ASP.NET Core для розробки серверної частини за допомогою мови C#, фреймворк Angular з використанням мови TypeSript для frontend частини веб-застосунку для взаємодії з користувачем. Для частини системи, яка являє собою пристрій який призначений для безпосереднього моніторингу даних з засобів обліку був використаний мікроконтролер ESP32, з сумісною відеокамерою. Для того щоб забезпечити постійний та надійний доступ до системи, було вирішено використати деякі з рішень. що пропонує хмарна платформа Microsoft Azure.

Передбачається, що система призначена для використання кінцевими користувачами які мають на меті контролювати і аналізувати споживання ресурсів шляхом зняття показань з засобів обліку. Тому однією з важливих речей на які необхідно було зробити акцент при розробці це зручність та простота використання.

1 АНАЛІЗ ВИМОГ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Загальний огляд

На ринку пропонується широка лінія так званих «розумних» засобів обліку в купі з програмним забезпеченням сумісним з такими засобами. Усі вони постулюють одні і ті самі переваги, основною є сама автоматизація рутинної задачі по обходу засобів обліку, контроль стану таких засобів та мереж куди вони встановлені та віддалений моніторинг показань таких засобів. Також, разом з такими «розумними» пристроями часто пропонується пакет програмного забезпечення у вигляди веб-застосунку, мобільного застосунку, або програмного рішення для персонального комп’ютера. В рамках дипломного проєкту пропонується альтернативне бачення таких пристроїв, основна перевага якого це можливість роботи з уже існуючими та встановленими комунікаціями. Для більш конкретного опису необхідно розглянути переваги та недоліки деяких продуктів що є аналогами дипломного проєкту окремо, а на основі отриманих даних сформулювати вимоги до системи моніторингу.

1.2 Аналіз існуючих аналогів

При аналізі існуючих на ринку аналогів необхідно безпристрасно підходити до кожного з варіантів для максимально об’єктивної оцінки. Під час аналізу вимог шляхом короткого огляду ринку стало зрозуміло на які фактори необхідно звернути увагу, адже вони є визначальними при виборі того чи іншого продукту.

Такими факторами є:

— вартість;

— масштабованість, тобто можливість підключати декілька пристроїв для обліку;

— доступ до зібраних даних, тобто дані доступні лише в локальній мережі чи вони доступні через Інтернет;

— можливість бездротового підключення до мережі;

— можливість використання з різними комунікаційними мережами;
— складність налаштування;
— необхідність додаткового обладнання;
— зручність програмного забезпечення;
— необхідність заміни існуючих комунікацій.

1.2.1 Система UniPing RS-485

З назви даного приладу для дистанційного моніторингу очевидно, що дані для взаємодії з передаються через інтерфейс RS-485. Дані передаються до спеціального приладу що виступає в ролі хаба, а вже сам хаб безпосередньо передає дані на веб-застосунок, який розгортується на користувацькій машині, до нього можна отримати доступ через локальну мережу. Хаб може взаємодіяти з декількома приладами обліку. Вся система в зборі є доволі громіздкою за рахунок необхідності з'єднання пристроїв в мережу і самого розміру цих пристроїв. Дані пристрої не можуть працювати в автономному режимі, для них потрібна окреме джерело живлення. В загальному, дану систему можна назвати найгіршою серед тих, що були розглянуті.

Переваги:

— можливість використовувати декілька приладів обліку;
— низька вартість.

Недоліки:

— незручне програмне забезпечення для доступу до даних;
— необхідна заміна штатних засобів обліку;
— автоматизація моніторингу можлива лише для електромереж;
— небхідність дротового підключення до мережі;
— необхідне додаткове обладнання у вигляді хабу;
— неможливість доступу до даних через Інтернет;
— складне налаштування системи.

1.2.2 Система Ecoisme

Дана система моніторингу являє собою прилад що приєднується до входу в електромережу будинку, підключається до домашнього Wi-Fi і збирає дані в реальному часі. На основі зібраних та проаналізованих даних можна побачити повну картину споживання електричної енергії, для цього використовується спеціальний мобільний додаток. На даний момент це одне із найпопулярніших рішень для вирішення задачі моніторингу засобів обліку родом з Великобританії.

Переваги:

— віддалений доступ до даних;
— зручний доступ до даних;
— можливість приєднання до Wi-Fi;
— не потрібно заміняти існуючі прилади обліку.

Недоліки:

— можуть бути складнощі при встановленні;
— можливість слідкувати лише за спожитою електроенергією;
— відсутність веб-версії додатку для перегляду даних;
— висока вартість.

1.2.3 Система Smart Mac

Це український стартап що являє собою сімейство розумних лічильників які здатні контролювати споживання як електроенергії, так і води, газу, тепла, вологість повітря, температуру, тощо. Зібрані дані зручно представлені в графічному вигляді у веб-застосунку.

Переваги:

— віддалений доступ до даних;
— можливість слідкувати за будь-якими засобами обліку;
— зручний доступ до даних;
— відносно невелика ціна.

Недоліки:

— необхідність заміни існуючих приладів.

Для того щоб отримані дані були більш наочними, в таблиці 1.1 наведено всі переваги кожного із трьох рішень що було описано вище.

Таблиця 1.1 — Порівняння систем для обліку

Фактор	Назва аналогів		
	UniPing RS-485	Ecoisme	Smart Mac
вартість	-	-	+
масштабованість	+	-	+
доступ до зібраних даних	-	+	+
можливість використання з різними комунікаційними мережах;	-	-	+
складність налаштування	+	+	+
необхідність додаткового обладнання	+	+	+
зручність програмного забезпечення;	-	+	+
необхідність заміни існуючих комунікацій.	+	+	+
можливість підключення до бездротової мережі	-	+	+

1.3 Вимоги до системи моніторингу

Аналізування вимог для нової чи зміненої системи це один з найважливіших етапів в розробці програмного забезпечення, саме по відповідності кінцевого продукту вимогам можна судити про якість виконаної роботи і чи була вона

взагалі виконана. Також, без прописаних вимог неможливо адекватно побудувати процес тестування та приймання в експлуатацію нової версії програмного забезпечення. Вимоги які мають бути сформульовані для будь якого програмного забезпечення можна описати як деяка сукупність характеристик і властивостей системи які притаманні даному продукту і роблять його корисним для власника, розробника та користувача. Зазвичай вимоги поділяють на функціональні и не функціональні[1]. Перші відповідають на питання що робить система, які задачі вона вирішує і яким чином це відбувається. Не функціональні мають на меті пояснити критерії для оцінки якості побудованого програмного забезпечення. Вони відповідають на питання якою система має бути. Наприклад, саме тут мають бути сформульовані певні обмеження для апаратних інтерфейсів з якими може взаємодіяти програмне забезпечення, опис платформи яка необхідна для запуску і подальшої роботи, вимоги щодо масштабування, надійності продуктивності та інші.

Функціональні вимоги часто представляють у вигляді так званої діаграми прецедентів зображеної на рисунку 1.1. Це дозволяє стисло та у графічному вигляді зобразити можливі дії які можуть бути здійснені користувачами з різними ролями.

З діаграми очевидно, що система має два види користувачів, адміністратора та звичайного користувача. Адміністратору доступні усі дії які тільки можуть бути доступні для звичайного користувача, а також можливість редагувати права будь-яких користувачів. RBAC[2] система доступу є практично, стандартом в світі сучасних веб-застосунків, вона дозволяє давати користувачам доступ лише до тих ресурсів, які необхідні для виконання їх задач, що гарантує безпеку даних у системі.

Передбачається, що головний функціонал веб-застосунку для користувачів це дії по візуалізації та експорт даних що були отримані з пристроїв встановлених на засоби обліку. Візуалізація представляє собою можливість побудови різноманітних графіків у вигляді: стовбчастих діаграм, кругових діаграм, кільце-

вих діаграм, тощо. Також, необхідно зазначити можливість агрегації даних за різні проміжки часу для подальшої роботи. Це дозволяє аналізувати дані про споживання за різні періоди часу.

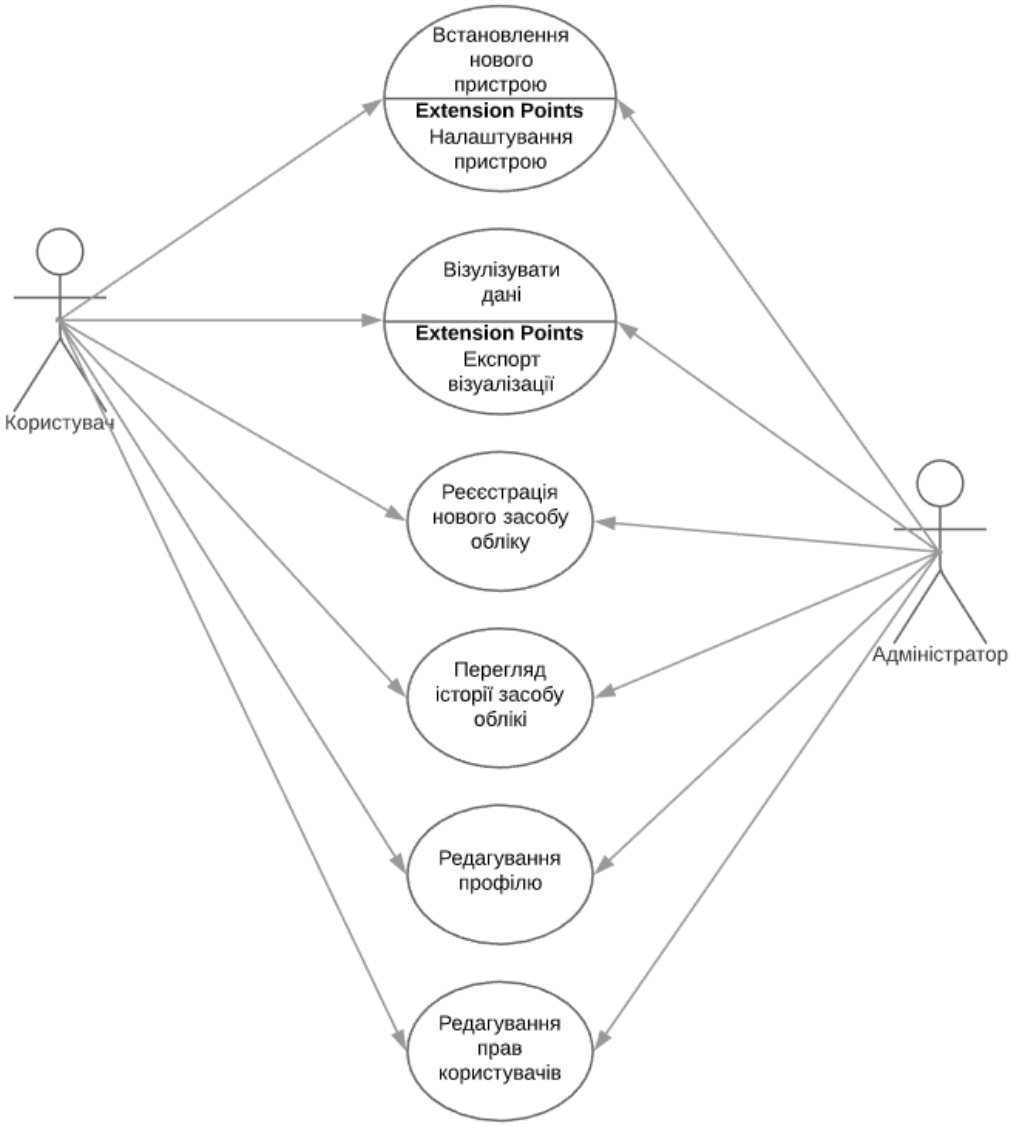

Рисунок 1.1 — Діаграма прецедентів

Експорт даних відбувається за рахунок генерації файлу у форматі pdf з обраною інформацією. Система має давати можливість доступу до усіх даних отриманиї від буд-якого зареєстрованого користувачем пристрою, за це відповідає функція перегляд історії.

Також, важливою функцією для користувача є можливість встановлення необмеженої кількості нових пристроїв для моніторингу та їх конфігурація.

Кожен користувач повинен мати свою приватну інформацію у вигляді налаштувань зовнішнього вигляду веб-застосунку, або своїх персональних даних, наприклад ім’я та адреса електронної пошти. У користувача має бути доступ до своєї приватної інформації для її перегляду і коригування у разі необхідності.

Як уже було сказано вище, не функціональні вимоги до системи визначають те якою система є. Такими вимогами є:

— система має бути завжди доступна;
— мати можливість для подальшого масштабування;
— мати зручний інтерфейс користувача;
— наявність авторизації;
— взаємодія між підсистемами має відбуватись через HTTP[3];
— має забезпечуватись адекватний рівень продуктивності.

1.4 Висновки до розділу

В даному розділі описано необхідні вимоги та аспекти системи на які необхідно звернути увагу в процесі розробки системи. Зважаючи на все вищесказане, перш за все необхідно забезпечити можливість використання пристрою з різноманітною існуючою інфраструктурою, а також максимально зменшити вартість такого пристрою. Що стосується користувацького досвіду, тут важливим елементом є доступ до даних з будь-якої точки світу та зручність програмного забезпечення що поставляється разом з пристроями для обліку і моніторингу.

Після аналізу усіх трьох систем стало очевидним, що кожна з них має певний набір критичних недоліків. Найбільш поширеним недоліком є те що кожен з варіантів потребує заміни існуючої інфраструктури що тягне за собою додаткові складнощі і затрати у вигляді ускладнення установки та повірки нового обладнання. Критичним мінусом також є сильна спеціалізація більшості з розглянутих систем, так, UniPing та Ecoisme можуть займатись моніторингом лише електричних мереж.

2 ТЕХНОЛОГІЇ І ПІДХОДИ ВИКОРИСТАНІ ПРИ РОЗРОБЦІ

2.1 Сервісно-орієнтована архітектура системи

SOA (Service oriented architecture) або сервісно-орієнтована архітектура це шаблон для проєктування програмного забезпечення, ідея якого в використанні розподілених та слабко пов’язаних компонентів, що можуть бути замінені, у всіх складових системи мають бути стандартизовані інтерфейси для взаємодії одним з одним за певними протоколами. Інтерфейси які використовують сервіси за своєю сутністю чітко визначені і незалежні один від одного, завдяки цьому досягається достатній рівень гнучкості і модульності, адже будь який модуль системи може бути замінений якщо він реалізує заданий інтерфейс.

Необхідно перерахувати базові принципи, які мають зберігатись при розробці системи згідно заданому шаблону[4]:

— відсутність стану;
— слабка зв’язність;
— повторне використання;
— стандартизація;
— відкритість;
— модульність;
— гранулярність.

Майже усі перераховані принципи інтуїтивно зрозумілі та доповнюють одне одного. Так, слабка зв’язність має на увазі невелику кількість залежностей між різними складовими системами. Такий підхід дає можливість повторного використання сервісу іншими підсистемами. Також однією з переваг при невеликій кількості залежностей між сервісами є можливість легко замінити один сервіс іншим, але це неможливо при відсутності певних стандартів якими розробники керуються під час проєктування і розроблення систем. Модульність означає, що кожен сервіс реалізує певні модулі бізнес-логіки, вони є частиною бізнес процесу і можуть бути замінені чи використані повторно, кожен модуль має відповідати

за одну функцію. Відсутність стану у сервісу означає, що сервіси не мають зберігати будь-яку інформацію про поточну сесію при роботі з клієнтом, такий підхід дозволяє не орієнтуватись на те що з клієнтом буде підтримуватись зв'язок довгий період часу. Відкритість – для того щоб повторно використати сервіс, він має бути зрозумілий і легко впізнаваний. При проєктуванні системи, кожен сервіс має брати на себе певну функцію в цілому бізнес процесі, це і є гранулярністю сервісів.

2.2 Принципи SOLID

Принципи SOLID це абревіатура, яка означає п'ять принципів яких необхідно притримуватись для написання якісного програмного забезпечення за допомогою об'єктно-орієнтовного програмування[5].

Буква S означає single responsibility principle (принцип єдиної відповідальності) – кожен компонент має відповідати лише за одну частину функціональності. Даний принцип має на меті превентивно прибрати дублювання коду, зміну уже існуючого коду, та зменшення складності подальшої підтримки і тестування.

Буква O означає open/closed principle (принцип відкритості/закритості) – кожен клас має бути закритим для змін, але відкритим для розширення. По суті, мається на увазі, що розширення функціоналу класу має здійснюватися за допомогою успадкування, проте в такому випадку кодова база росте, а значить підтримка такого коду ускладнюється.

Буква L означає Liskov substitution principle (принцип підстановки Лісков) – має на увазі, що будь-який об'єкт типу можна замінити на об'єкт нащадок даного типу і поведінка програми не зміниться.

Буква I означає interface segregation principle (принцип розділення інтерфейсів) – необхідно створювати багато вузько-спеціалізованих інтерфейсів на відміну від створення одного великого. Це дозволяє розділяти відповідальність по різним частинам програми.

Буква D означає dependency inversion principle (принцип інверсії залежностей) – модулі вищих рівнів мають залежати від абстракцій так само як і модулі нижчих рівнів. Абстракції не мають залежати від деталей.

2.3 Шаблон MVC

Архітектурний шаблон MVC – model view controller є класичним шаблоном для проєктування веб-застосунківву, який широко використовується для розробки програмного забезпечення. Цей шаблон є стандартним при роботі з такими фреймворками як Angular та ASP.NET Core, тому його використано в дипломному проєкті. Ідея в тому щоб розділити відповідальність за різні аспекти роботи програмного забезпечення на різні компоненти. Схема взаємодії цих трьох складових зображена на рисунку 2.1.

Рисунок 2.1 — Діаграма шаблону

В даному шаблоні модель це центральний компонент, він являє собою структуру даних незалежну від користувацького інтерфейсу, він управляє даними, логікою та бізнес-правилам в застосунку.

Вигляд забезпечує відображення інформації для кінцевого користувача у різних виглядах, наприклад графік, таблиця тощо

Контролер потрібен щоб зв’язати два інших компоненти між собою, отримувати вхідні дані, проводити з ними маніпуляції, передавати їх у модель і формувати відповідь.

2.4 Технології для розробки серверної частини веб-застосунку

.NET Core це безкоштовна платформа для розробки програмного забезпечення будь-якої складності з відкритим кодом. Головною перевагою даної платформи є те, що вона підтримується компанією Microsoft а також те, що програмного забезпечення на її основі може бути запущене на будь-якій з трьох найпопулярніших операційних систем: Windows, Linux, MacOs.

Дана платформа була обрана тому що вона має ряд переваг перед старшою платформою .NET Framework[6]. Спільнотою та самою компанією Microsoft рекомендується використовувати .NET Core у випадках коли:

— необхідно створити кросплатформне рішення;

— система має мікросервісну архітектуру;

— при використанні Docker[7] контейнерів;

— якщо масштабованість та висока продуктивність важливі.

Так, виходячи з вимог що були сформульовані в першому розділі, стає зрозуміло що система моніторингу підпадає під майже усі рекомендації. Більш того, система має бути розгорнута на хмарній платформі Microsoft Azure використовуючи PaaS рішення, в свою чергу вони підтримують роботу лише з платформою .NET Core.

ASP.NET Core є новим фреймворком для розробки сучасних веб-застосунків. Він працює на базі платформи .NET Core, завдяки чому веб-застосунки побудовані на його базі є крос-платформними і можуть бути розгорнуті на будь-якій машині. Цей фреймворк також з відкритим кодом, завдяки чому він активно розробляється спільнотою і відкритий до змін. Фреймворк має багато переваг що прискорюють та облегшують розробку програмного забезпечення:

— вбудована система введення залежностей;

— висока продуктивність;

— багато опцій для розгортування;

— зручне тестування;

— може використовуватись як для серверу, так і для клієнтського додатку.

В дипломному проєкті, фреймворк використано для розробки backend веб-застосунку для взаємодії з користувачами.

Будь-яка сучасна система має мати можливість для авторизації користувача. Ця вимога може бути продиктована різними міркуваннями, наприклад персоналізація досвіду роботи з системою чи захист чутливих користувацьких даних від несанкціонованого доступу.

ASP.NET Core Identity, це спеціальний nuget[8] пакет що є системою авторизації яка дозволяє додати функцію входу у веб-застосунку. Перевага в тому що даний пакет може бути налаштований для роботи з постійним сховищем таким як база даних чи Azure Table Storage, так і для роботи з зовнішніми провайдерами такими як Facebook, Twitter, Google тощо.

Для реалізації авторизації було обрано підхід з JWT[9] токенами. Принцип роботи простий, клієнт робить запит на сервер, отримує токен, потім прикладає в заголовок до кожного запита на сервер цей токен, завдяки чому на сервері є можливість перевірити особистість користувача від імені якого зроблено запит.

Авторизація за допомогою JWT токенів також має використовуватись при мікросервісній архітектурі, адже дає можливість авторизуватись одночасно на багатьох різних сервісах без необхідності обміну даними про поточну користувацьку сесію.

2.5 Шаблон Repository unit of work

Передбачається, що система має бути гнучкою і відкритою для змін, які можна впровадити відносно легко. Тому, для доступу до джерела даних було обрано спеціальний шаблон, який дозволить досягнути цих цілей.

Шаблон Repository unit of work є комбінацією двох шаблонів: Unit of work та Repository. Шаблон Repository[10] необхідний для того щоб бути додатковим шаром абстракції між шаром бізнес-логіки та шаром доступу до даних використовуючи деякий інтерфейс. UML діаграма шаблону представлена на рисунку 2.2. Перевага такого підходу в тому що він дозволяє відв'язати бізнес логіку від безпосередньої взаємодії з джерелом даних, дає можливість інкапсулювати запити до джерела даних та полегшує майбутнє тестування.

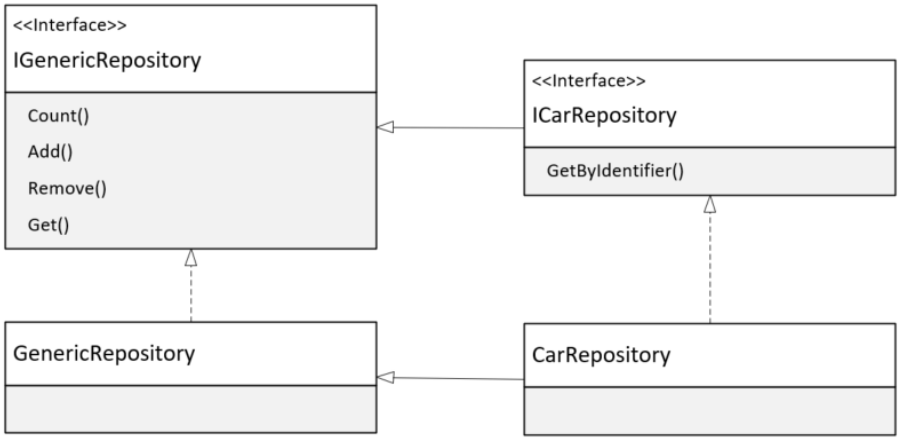

Рисунок 2.2 — Діаграма шаблону Repository

Шаблон Unit of work покликаний надавати можливість зберігати зміни в сховищі даних в рамках однієї транзакції. У купі з використанням шаблону repository, полегшує доступ до репозиторіїв, тому що всі вони зібрані в одному місці. В свою чергу, це надає можливість для використання усіма репозиторіями одного підключення до БД та лінивої ініціалізації репозиторіїв.

2.6 Робота з даними

Практично будь-який додаток має зберігати дані для подальшої роботи з цими даними. Загальний підхід до збереження чітких структурованих даних використовувати в якості сховища реляційну базу даних з відповідною СУБД . В даному випадку використана СУБД MSSQL.

Так як SQL запити зазвичай об’ємні та складні, часто використовуються ORM для зручної роботи з базою даних, більш того деякі ORM дозволяються працювати з БД в об’єктно-орієнтованому стилі. В роботі був використаний Entity Framework Core в якості такої ORM. Ця ORM була обрана через те, що вона повністю сумісна з LINQ запитами, є дуже популярною, добре документованою та підтримує роботи з будь-якою відомою реляційною СУБД.

Для зберігання статичних даних – зображень використане спеціальне хмарне сховище Azure Blob Storage. Його основною перевагою є оптимізованість для зберігання великих неструктурованих масивів даних, є дешевим і таким, що добре масштабується. Вибір пав на нього через орієнтацію на використання хмарної платформи та можливість тісної інтеграції з іншими сервісами. Рекомендується використовувати це сховище у таких випадках:

— відображення зображень чи документів прямо у браузері;
— зберігання файлів для розподіленого доступу;
— потокова передача відео чи звуку;
— записування і зберігання логів;
— архівація чи звичайне зберігання даних.

В випадку дипломного проєкту, сценарій використання в якості сховища для статичного контенту підходить ідеально під ці рекомендації.

2.7 Хмарна платформа Microsoft Azure

Microsoft Azure це одна з найпопулярніших на теперішній день хмарних платформ, яка поступово завойовує ринок хмарних обчислень. Як і будь-яка інша хмарна платформа вона може виступати в якості хостинга для різноманітних додатків, а також надає доступ до великої кількості різних хмарних сервісів.

В цілому, платформа надає послуги IaaS[11] (інфраструктура як сервіс) та PaaS[12] (платформа як сервіс). В даній роботі використані PaaS-послуги даної платформи для розгортання додатку та для зберігання і обробки даних.

2.8 Технології використані для безсерверного обробника фотографій

При виборі технологій для реалізації підсистеми для оброблення фото, необхідно було знайти спосіб зберігати зображення і в подальшому оброблювати ці фото. Тому, для збереження було обрано Blob Storage, а сам додаток було вирішено зробити безсерверним на базі Azure Functions.

Azure Functions це подійно-орієнтована платформа безсерверних обчислень, що надається платформою Azure. Головними особливостями є глибока інтеграція з іншими сервісами, що надаються хмарною платформою, автоматичне та гнучке масштабування, яке не потребує додаткового втручання розробника та модель розробки що базується на подіях, що є зручною можливостю для інтеграції с іншими сервісами. В цілому, використання цієї платформи дозволяє зняти з програміста відповідальність за налаштування інфраструктури для запуску програмного забезпечення.

Таким чином, завдяки можливості реакції на події, що згенеровані Blob Storage, Azure Functions є ідеальним вибором для реалізації задуманої підсистеми для роботи з фотографіями.

Для розпізнавання тексту на фото було вирішено використовувати сторонній сервіс для зменшення загальної складності проєкту. Такий сервіс надається платформою Azure, у порівнянні з іншими сторонніми рішеннями має адекватну вартість, а тому було вирішено використовувати Azure Computer Vision API.

Azure Computer Vision API – це API що надається хмарною платформою Azure та дозволяє використовувати технології комп’ютерного зору для розпізнавання зображень. Використання штучного інтелекту в сучасному світі є дуже потужним інструментом і такі сервіси значно полегшують розробку додатків. В даній роботі сервіс використовується для витягання тексту з зображень засобів обліку.

2.9 Технології використані при розробці клієнтської частини

Для розробки клієнтської частини веб-застосунку для взаємодії з користувачами треба було обрати технологію що дозволить побудувати сучасний SPA[13] веб-застосунок.

Angular – це фреймворк та платформа для розробки ефективних та складних SPA (single page application) застосунків. На даний момент, це один із найпопулярніших JavaScript фреймворків для розробки клієнтської частини веб-застосунків. Так як Angular широкорозповсюджений, для нього існує багато різноманітних бібліотек для побудови користувацького інтерфейсу та різних компонентів. Основні можливості які позитивно відрізняють Angular від конкурентів:

— компонентна структура веб-застосунку;
— двостороннє зв’язування;
— вбудоване впровадження залежностей;
— вбудована маршрутизація;
— можливість вести розробку мовою TypeScript.

Розробка ведеться мовою TypeScript що є надбудовою, яка розширює можливості JavaScript, зокрема мова підтримує повноцінні класи та явне визначення типів. Завдяки цьому факту, загальна структура проєкту стає більш прозорою, подальша підтримка простішою, а розробникам, що звикли до статичної типізації стає легше розробляти функціонал для проєкту мовою TypeScript.

Так як побудова зручного користувацького інтерфейсу це важлива частина досвіду що отримає користувач при роботі з додатком, необхідно було прийняти рішення щодо принципів побудови користувацького інтерфейсу.

Найпоширеніше рішення це використати бібліотеку компонентів таку як Bootstrap, Angular material чи PrimeNG та кастомізувати функціонал, що надається ними відповідно до вимог до конкретного елементу інтерфейсу. Кастомізація елементів інтерфейсу проводиться шляхом додавання CSS стилів та зміни HTML розмітки.

2.10 Технології для генератора фотографій на базі мікроконтролера.

В якості мікроконтролеру було обрано відомий контролер з назвою ESP32[13], більш точно, його модифікацію ESP32-Cam що містить в собі роз'єм для камери та карти пам'яті, мікроконтролер зображено на рисунку 2.3.

Рисунок 2.3 — Мікроконтролер ESP32-Cam

Цей контролер є новою, покращеною версією контролеру ESP8266. Він є більш потужним та має більше пам'яті. Головною особливістю даного мікроконтролера є встановлений Wi-Fi модуль, що дозволяє приєднатись до існуючої мережу, або створити нову.

Контролер обрано так як це одне із найдоступніших рішень які дозволяють одночасно працювати з мережею та робити фотографії. Більш того, контролер має роз'єм для карти пам'яті формату MicroSD що позитивно відображається на спектрі можливих застосувань.

В якості камери було обрано OV2640 так як вона є однією з найпоширеніших при роботі з даним мікроконтролером, тому існує чимало бібліотек для роботи з нею.

Одна із вимог до системи це автономність та відносно невеликий розмір, тому необхідно мати автономне джерело живлення для мікроконтролеру. В якості такого можуть виступати будь-які елементи живлення. В ході пошуку найкращого варіанту, було вирішено використати батарейний бокс для 4 батарейок типу AAA. Таким чином, буде існувати можливість використовувати як звичайні батарейки, так і акумулятори. Так як стандартна напруга в батарейок 1.5 вольта, а мікроконтролер споживає 5 вольт, необхідно використати DC/DC конвертер для пониження напруги до номінальних значень.

Так як контролер підтримує режим сну, така конфігурація цілком дозволить живити контролер доволі тривалий проміжок час без заміни елементів живлення, в той же час, саме джерело живлення не займе багато місця.

2.11 Висновки до розділу

В даному розділі було розглянуто та обґрунтовано вибір технологій та підходів що використані при проєктуванні та подальшій розробці усіх підсистем, починаючи веб-застосунком і закінчуючи апаратною частиною.

Було вирішено використовувати велику кількість хмарних технологій, адже вони мають ряд беззаперечних переваг перед традиційними рішеннями. Наприклад, забезпечується можливість для подальшого масштабування та зручна робота з іншими хмарними сервісами що полегшує і прискорює процес і якість розробки.

Звичайно, обрані рішення мають і недоліки. Так, деякі обрані рішення, такі як сховище статичного контенту, сервіс розпізнавання тексту та сервіс для розміщення реляційної бази даних є завжди платними, а сервіси для розміщення безсерверного додатку та веб-застосунку є платними в випадку необхідності більшої кількості ресурсів. Також, можна назвати недоліком і доволі велику кількість різних технологій, що суттєво збільшує поріг входу в проєкт для розробників що можуть займатися подальшою розробкою функціональності.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Загальний огляд

На основі аналізу аналогічних існуючих систем сформульованих вимог було проведено декомпозицію комплексної задачі на менші підзадачі, а усю функціональність було розбито між декількома підсистемами. Розглянемо загальну схему системи, що зображена на рисунку 3.1.

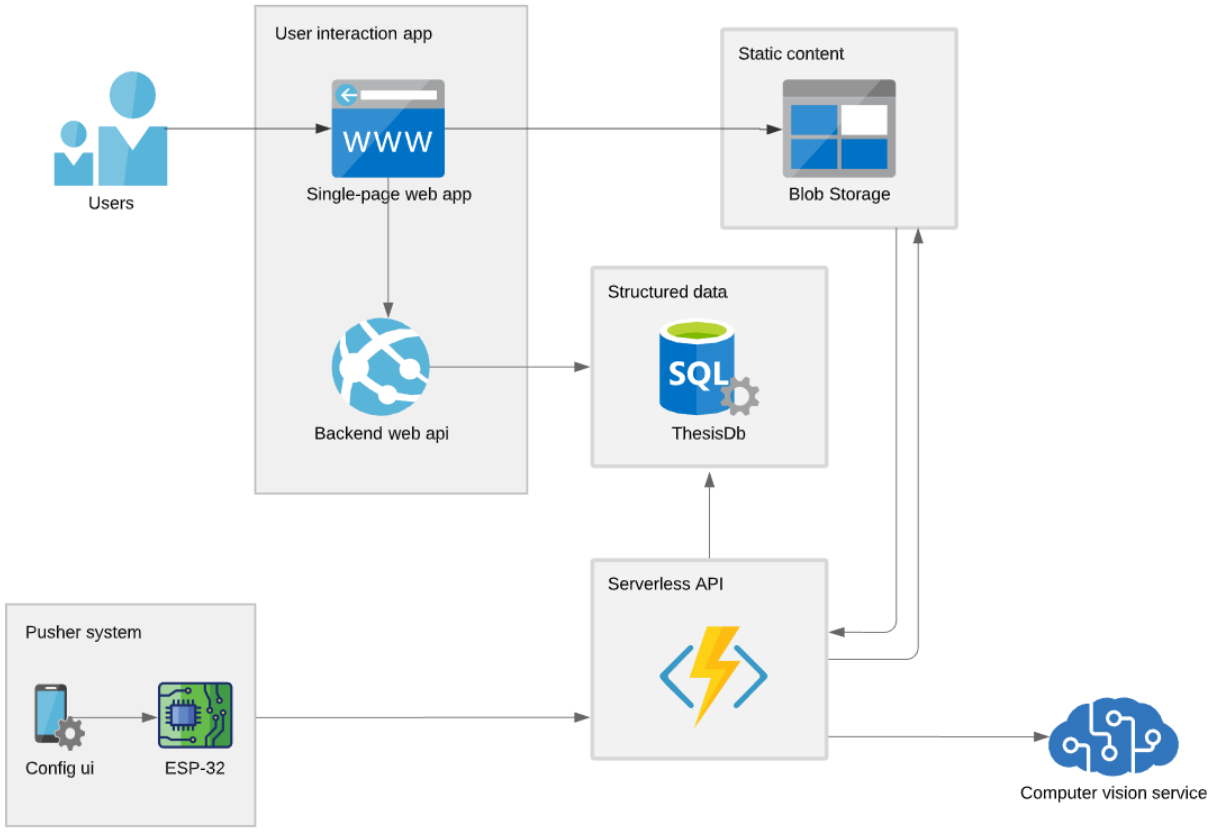

Рисунок 3.1 — Загальна схема системи

З рисунку стає очевидно що загальна система для моніторингу засобів обліку складається з трьох менших підсистем, а саме:

— генератор фотографій, тобто мікроконтролер який безпосередньо встановлюється на засіб обліку;

— обробник фотографій, тобто безсерверне API побудоване за допомогою Azure Functions;

— веб-застосунок для взаємодії з користувачами.

Всі взаємодії між сервісами відбувається за стандартизованими контрактами, по найпоширенішому протоколу HTTP[3]. Також для більш зручної роботи зі сховищем статичного контенту також використовуються події яке воно генерує при взаємодії з ним.

Підсистема генератор фотографій, яка містить в собі мікроконтролер відповідальна за взаємодію з засобом обліку на який встановлюється сама підсистема. Обраний мікроконтролер доволі потужний, а тому здатен запускати в собі повноцінний веб-сервер і загалом взаємодіяти через мережу. Веб-сервер піднятий на генераторі фотографій має слугувати приймачем конфігураційних запитів для налаштування поточної Wi-Fi мережі, інтервалу для відправки даних, тощо. За допомогою камери, яка встановлена на мікроконтролері, він робить фотографію один раз у визначений проміжок часу та відправляє її для подальшої обробки на обробник фотографій.

Обробник фотографій побудований на основі Azure Functions і відповідальний за отримання фотографій від підсистеми генератор фотографій , їх збереження в сховище статичного контенту, реагування на події від сховища статичного контенту, розпізнавання тексту на фотографіях за допомогою стороннього сервісу комп’ютерного зору та збереження отриманих результатів в реляційну базу даних.

Додаток для взаємодії з користувачами складається з двох менших підсистем, а саме frontend частини та backend частини. Frontend відповідає за безпосереднє візуальне представлення даних користувачеві за допомогою графіків і таблиць, а backend обслуговує frontend і виконує функції авторизації користувача, взаємодії з базою даних та агрегації даних.

3.2 Підсистема генератор фотографій

Підсистема генератор фотографій представляє собою пристрій, який підключається до існуючого засобу обліку, та робить його фотографії через певні проміжки часу та відправляє їх на обробник фотографій. Пристрій побудовано

на базі мікроконтролера ESP32 з камерою OV2640. Для живлення використовується 4 акумулятори типу AAA. Пристрій підключений до джерела енергії за допомогою понижуючого стабілізатору. Схема підключення програматора, стабілізатора та аккумулятору до ESP32-CAM представлена на рисунку 3.2.

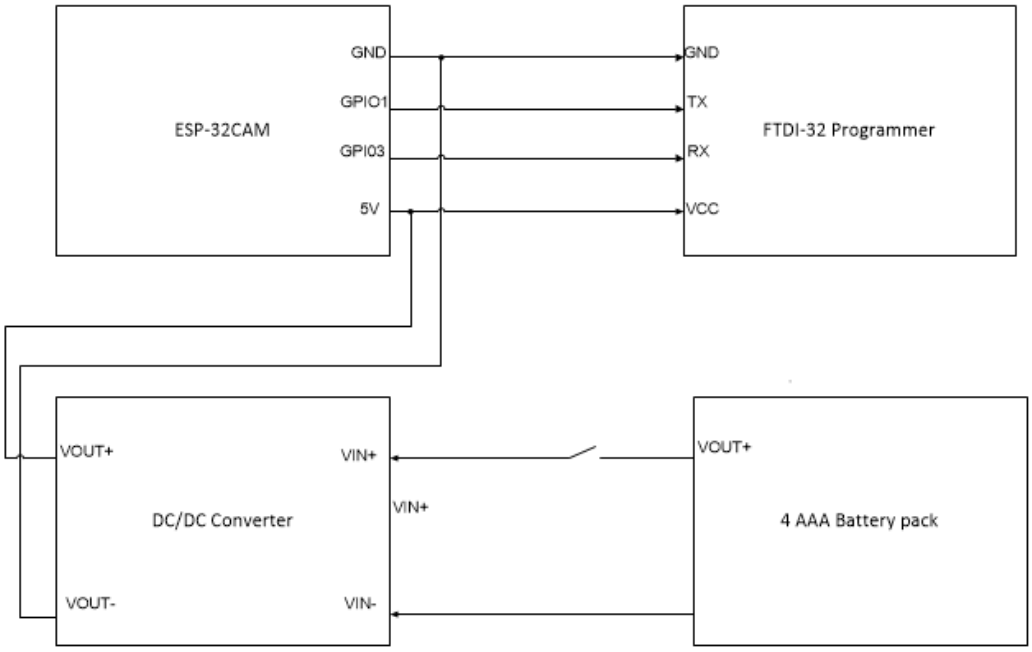

Рисунок 3.2 – Схема підключення до ESP32-CAM

У ході розробки даної системи було протестовано декілька бібліотек для генерації HTTP запитів та взаємодії з Wi-Fi мережею. Після тестування, було обрано стандартні бібліотеки, що надає компанія розробник мікроконтролера – «esp_http_client» та «esp_WiFi». На даний момент, в цілях спрощення і побудови так званого MVP дані для підключення до існуючої Wi-Fi мережі були прописані у виді константи безпосередньо у вихідному коді.

Камера використовується через спеціальний драйвер з відкритим вихідним кодом «esp_camera». Фото робиться шляхом захоплення даних з матриці, записі їх у буфер в оперативній пам’яті, тому якість зображення доволі обмежена.

Також, необхідно зауважити, що при розробці було використано режим глибокого сну для оптимізації споживання електричної енергії. Це досягається

шляхом відключення усіх частин мікроконтролера окрім ULP співпроцесору, таймеру реального часу та виходів для периферії GPIO. Така техніка дозволяє зменшити споживання з 200 мА до 0.15мА. ESP32 має декілька можливостей для переходу та виходу зі сну, в даному випадку використовується вбудований таймер, що дозволяє мікроконтролеру переходити у нього в інтервали між фотографуванням.

Вихідний код для мікроконтролера складається з одного файлу та містить в собі декілька методів, констант та змінних. А саме, константи що задають час інтервальних проміжків, дані для підключення до Wi-Fi мережі, та константи, які визначають виходи GPIO. Змінні необхідні для збереження поточного стану підключення до мережі.

Методи, які містяться в коді:

— print_wakeup_reason – потрібен для виведення типу пробудження через COM порт, до якого підключений програматор;

— setup – метод який викликається при пробудженні мікроконтролеру, тут відбувається ініціалізація камери та відправка зображення.;

— init_wifi – необхідний для ініціалізації Wi-Fi модулю та підключення до мережі;

— _http_event_handler – використовується для реагування на відповідь, яка була отримана в результаті HTTP запиту;

— take_send_photo – використовується для фотографування та подальшої роботи з отриманим зображенням.

3.3 Підсистема обробник фотографій

Обробник фотографій представляє з себе безсерверне API яке побудовано на базі технології Azure Functions. Використання цієї технології дозволило тісно інтегруватись з іншим сервісом, який призначено для зберігання статичного контенту – Blob storage. За допомогою спеціальних трігерів, які називаються Blob-Trigger, розроблені функції мають можливість реагувати на події які генеруються

сховищем. В даному випадку, функція реагує на подію завантаження нової фотографії у сховище. Такий підхід має велику перевагу, адже дає можливість для подійно-орієнтованого підходу при побудові архітектури системи. Його переваги було розглянуто у розділі 2.

3.3.1 Процес отримання фото

Для розуміння алгоритму роботи даної системи при отриманні фото розглянемо схему на рисунку 3.3, на ній зображено першу з двох частин сценарію роботи з отриманим зображенням. За прийом фотографії відповідає функція під назвою UploadPictureFunction.

Зі схеми стає очевидним сценарій обробки отриманого фото. Спочатку, функція отримує HTTP запит, який приходить від мікроконтролеру, далі проходить процес валідації отриманих даних з мікроконтролеру. Цей процес полягає в перевірці отриманого контенту на його довжину та наявність спеціального ідентифікатору пристрою, що прислав запит, адже якщо у ньому немає даних чи ідентифікатору для обробки, то немає сенсу продовжувати з ним роботу.

На наступному кроці за допомогою отриманого ідентифікатору пристрою, проходить перевірка на те чи зареєстрований пристрій чи ні. Якщо перевірка успішно пройдена і в системі є користувач, який зареєстрував даний пристрій, то система переходить до наступного етапу. Тепер, задача полягає в том щоб створити новий Blob[14] в сховищі та завантажити у нього отримане фото. Якщо дана процедура була виконана успішно, то система зберігає запис інформації про фото у базу даних, ця інформація включає у себе:

- посилання на фото у сховищі;
- ім’я фото;
- ідентифікатор пристрою, що зробив фото;
- час захоплення даних з пристрою обліку.

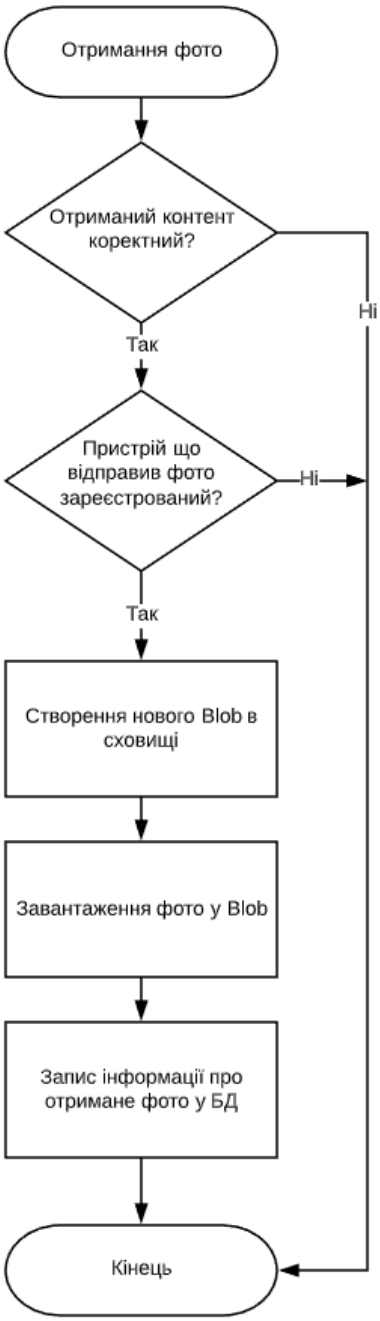

Рисунок 3.3 — Алгоритм прийому фотографій

3.3.2 Процес розпізнавання тексту на фото

Друга частину сценарію обробки фотографії зображена на рисунку 3.4. За нього відповідає функція з назвою RecognizePictureText. Даний сценарій починає працювати зразу після завершення минулого за допомогою події яка генерується BlobStorage[15] в момент коли фото завантажене у сховище. Після цього функція

з'єднується з сервісом комп'ютерного бачення і авторизується там. Якщо з'єднання було успішним, функція робить запит до цього сервісу і очікує у циклі допоки результат розпізнавання зображення не стане доступний. Після отримання результатів, за допомогою регулярних виразів проходить пошук відповідного значення в залежності від типу пристрою. Якщо підрядок не було знайдено, то відповідний запис у базі даних помічено як некоректний і не буде більше опрацьований.

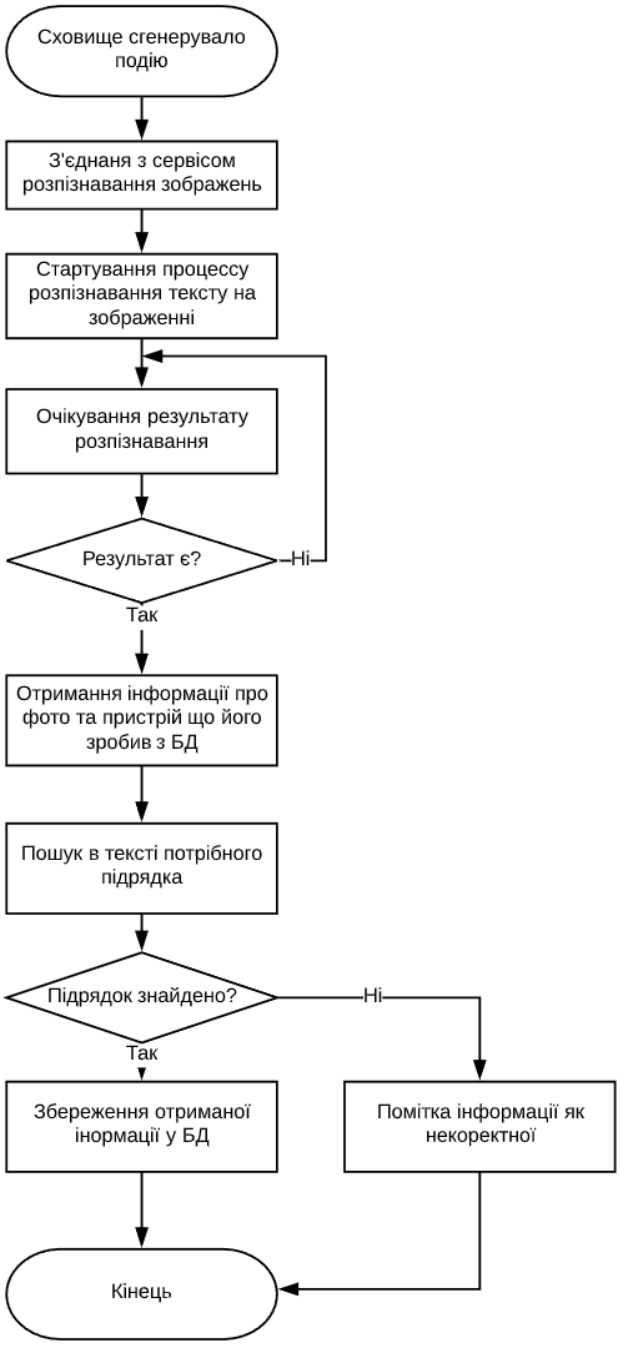

Рисунок 3.4 — Алгоритм розпізнавання потрібного значення

3.3.3 Структура проєкту

Підсистема обробник фотографій була виконана у вигляді окремого рішення у середовищі Visual Studio 19. Під час розробки необхідно було зберегти структуру достатньо щоб забезпечити просте впровадження подальших змін. Діаграма класів проєкту безсерверного backend зображена на рисунку 3.5.

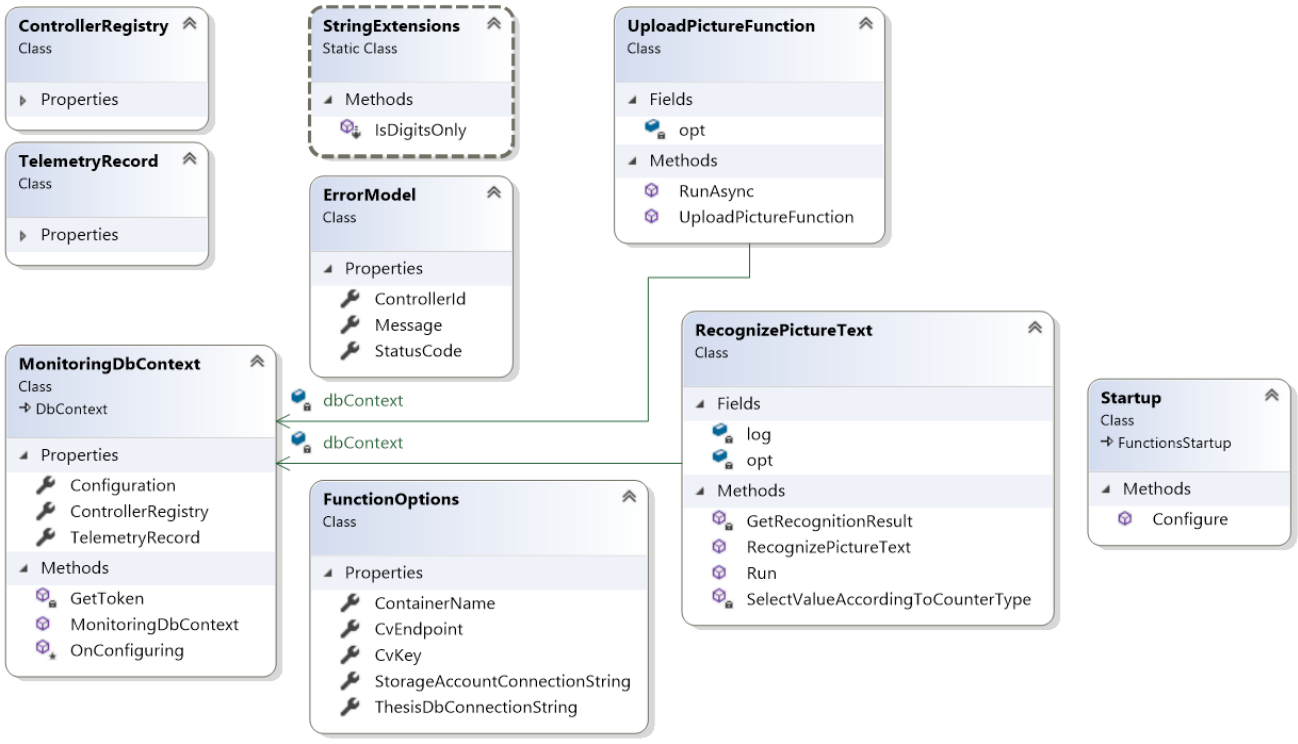

Рисунок 3.5 — Структура проєкту ServerlessApi

Контекст та моделі призначені для конфігурації та роботи з базою даних за допомогою EF Core. Структура бази даних розглянута у наступному розділі. Ця частина функціональності реалізована за допомогою трьох класів:

— ControllerRegistry – клас який є моделлю, яка відповідає таблиці ControllerRegistry в базі даних і повністю їй відповідає;

— TelemetryRecord – клас який є моделлю і повністю відповідає таблиці TelemetryRecord;

— MonitoringDbContext –клас, який є контекстом для роботи з базою даних за допомогою Entity Framework Core.

Клас StringExtenstions призначений для спеціальних методів розширення які дозволяють розширити функціональність уже існуючих типів без їх безпосередньої модифікації. В даному випадку, розроблено розширюючий метод IsDigitsOnly для типу String який дозволяє у зручний спосіб перевіряти чи усі символи у рядку є цифрами.

Клас ErrorModel являє собою структуровану відповідь, яку висилає функція у випадку будь-яких виключних ситуацій.

Клас FunctionOptions призначений для проекції файлу налаштувань за допомогою шаблону IOptions. Такий підхід дозволяє ізолювати файли налаштувань по різним класам та використовувати DI для передачі об'єкту класу з налаштуваннями у всі місця де він очікується.

RecognizePictureText – як було описано раніше, функція що відповідає за розпізнавання тексту на зображеннях. Містить 3 методи:

— Run – запускає функцію, відповідальний за взаємодію з сервісом комп'ютерного зору та збереження даних;

— SelectValueAccordingToCounterType – відповідає за пошук необхідного значення у розпізнаному тексті;

— GetRecognitionResult – відповідає за безпосередню взаємодію з сервісом комп'ютерного зору за допомогою спеціальної бібліотеки.

Клас UploadPictureFunction – містить у собі функцію, що відповідальна за процес прийому запитів від мікроконтролера. Має один метод Run, в якому і відбувається уся робота.

Клас Startup – вхідна точка в додаток, містить один метод Configure який необхідний для реєстрації усіх сервісів в IOC контейнері.

3.4 Веб-застосунок для взаємодії з користувачами

Веб-застосунок для взаємодії з користувачами складається з двох менших підсистем – backend побудований за допомогою фреймворку ASP.NET Core та

frontend побудований за допомогою фреймворку Angular. Таким чином, ця підсистема побудована за допомогою клієнт-серверної архітектури. В даній релізації fronend є тонким клієнтом, тобто забезпечує лише функції представлення даних, це дозволило зробити його максимально легким та швидкодійним. Backend відповідає за бізнес-логіку та за управління даними, тобто доступ і їх зберігання.

3.4.1 Структура бази даних

Для зберігання даних про користувачів, пристрої та дані що надсилаються цими пристроями необхідно було розробити структуру бази даних з усіма обов'язковими зв'язками між таблицями, що дозволило зручно користуватись базою даних. Для безпосередньої роботи з базою даних, використовувався EF Core, робота з ним розглянута у наступному розділі. Розглянемо схему бази даних зображену на кресленику ІТ61.240БАК.004 Д1.

На ній зображені такі таблиці:

— TelemetryRecord – призначена для зберігання інформації про поточний стан засобів обліку у вигляді розпізнаного числа і фотографії;

— ControllerRegistry – призначена для зберігання усіх зареєстрованих користувачами пристроях для моніторингу;

— AspNetUsers – призначена для зберігання даних про користувачів;

— AspNetRoles – призначена для зберігання ролей, які є в системі;

— AspNetRoleClaims – призначена для зберігання дозволів які є у певних ролей;

— AspNetUserRoles – призначена для збереження даних про те, які ролі є у користувача, являє собою розв'язочну таблицю між AspNetRoles та AspNetUsers;

— AspNetUserClaims – призначена для зберігання дозволів, що були надані користувачу;

— AspNetUserLogins – призначена для зберігання інформації про входи що були здійснені за допомогою стороннього провайдеру;

— AspNetUserTokens – призначена для зберігання JWT токенів в разі входу зовнішнього провайдеру;

— _EFMigrationsHistory – призначена для збереження даних про міграції, що були застосовані по відношенню до бази даних.

3.4.2 Серверна частина веб-застосунку

Backend веб-застосунку в основному працює з користувачами та записами у базі даних для записів про зареєстровані пристрої та дані отримані з цих пристроїв. Під час розробки усе рішення було розбито на два проєкти з метою розділення відповідальності між ними. Проєкт DataAccess відповідальний за доступ до даних, а проєкт MonitoringSpa відповідає за бізнес-логіку і представлення даних для frontend.

Розглянемо діаграму класів проєкту DataAccess зображену на кресленику ІТ61.240БАК.004 Д2. Усі класи було умовно розділено на декілька частин.

Доступ до даних був реалізований з використанням шаблону repository unit of work. Таким чином, це дозволило розділити доступ до даних через ще один рівень абстракції, це має декілька безперечних переваг. Головною є можливість проводити декілька операцій з базами даних в рамках однієї транзації.

Також для роботи з даними в проєкт були включені так звані міграції. Використання міграцій в даному проєкті дозволило спростити процес розгортки додатку так як уся необхідна структура створюється автоматично.

Core відповідає за класи, які необхідні для роботи з користувачами, їх обліковими записами та дозволами цих користувачів.

В блоці UnitOfWork знаходяться інтерфейс для UnitOfWork, його реалізація UnitOfWork, містить в собі усі репозиторії які використовуються для доступу до конкретних сутностей в базі даних, а також метод SaveChangesAsync, який завершує поточну транзакцію та зберігає зміни що знаходяться у контексті ApplicationDbContext. Також для інтеграції з ASP NET Identity був розроблений

HttpUnitOfWork, він є похідним від UnitOfWork та розширює функціонал шляхом роботи з поточним користувачем в HttpContext.

Контекст є частиною моделі яка вимагається EF Core та є своєрідною моделлю бази даних представленою в об'єктно-орієнтованому вигляді. Для представлення таблиць в базі даних, контекст містить поля спеціального узагальненого типу DbSet. Кожній таблиці відповідає свій DbSet з відповідною моделлю. Для того щоб автоматично відслідковувати всі зміни які відбуваються з бізнес-сутностями що реалізують інтерфейс IAuditableEntity, в контексті був перевизначений метод SaveChangesAsync.

Repositories призначені для репозиторіїв, для кожної моделі існує свій репозиторій. Це дозволило відв'язати доступ до джерела даних від конкретного підключення до нього.

Models містить моделі, які використовуються для взаємодії з базами даних. Для того щоб запевнитись що всі бізнес-сутності мають певний набір полів що включає унікальний ключ ідентифікатор та набір полів з інформацією про те, хто і коли створив чи змінив дану сутність було створено спеціальний інтерфейс IAuditableEntity та його реалізацію AuditableEntity. Для того щоб додати необхідний набір полів до бізнес-сутностей ці сутності наслідують AuditableEntity. Діаграму класів моделей представлено на рисунку 3.6. Моделі ApplicationRole та ApplicationUser не наслідуються від AuditableEntity, натомість, реалізують інтерфейс IAuditableEntity тому що ці моделі розширюють функціональність класів IdentityRole та IdentityUser відповідно. Ці два класи надаються пакетом ASP.NET Core Identity, що призначений для авторизації і управлінню користувачами.

Проєкт MonitoringSpa, як було сказано вище, відповідальний за прийом та обробку HTTP запитів, що приходять від frontend додатку відповідно до реалізованої бізнес логіки. Проєкт представляє з себе класичне REST[16] API. Для кращого розуміння структури даної підсистеми, розглянемо діаграму основних класів проєкту, що зображена на кресленику ІТ61.240БАК.004 Д3.

Секція Authorization містить спеціальні класи які необхідні для конфігурування та роботи з ASP.NET Core Identity. Розглянемо їх більш детально.

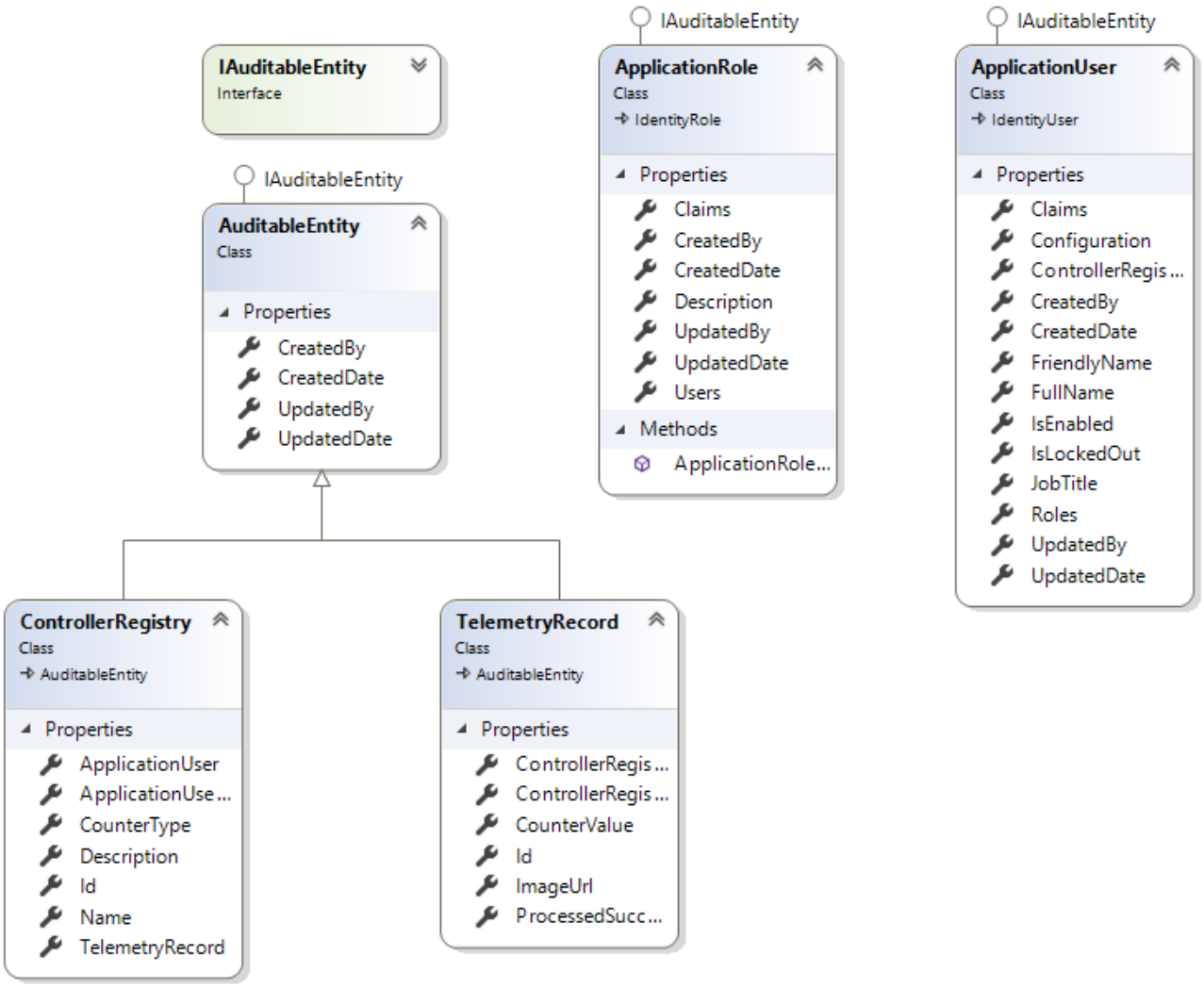

Рисунок 3.6 — Діаграма класів моделей

Клас Policies містить перелік можливих політик доступу до даних, наприклад дозвіл на переглядання усіх користувачів.

Клас ProfileService необхідний для інкапсуляції логіки роботи з користувачами, додатковий шар абстракції дозволив зменшити заплутаність коду в місцях, де відбувається робота з користувачами.

Класи, які містять у назві слово requirement працюють у зв’язці з відповідними класами handlers які спеціальними обробниками, в яких знаходиться логіка перевірки дозволів у користувача основуючись на класі requirement. Наприклад, лише користувач, у якого є дозвіл на операції перегляду користувачів, може їх переглядати.

Секція Controllers містить 4 контролери, які приймають усі HTTP запити. Кожна REST кінцева точка має обмеження, запит не буде оброблений якщо він не містить заголовка з необхідним JWT токеном. Виключенням є кінцеві точки призначені для реєстрації та входу користувача у систему. Контролер AccountController призначений для усіх операцій пов’язаних з користувачами:

— перегляд існуючих користувачів;
— перегляд ролей цих користувачів;
— створення нових користувачів;
— редагування існуючих користувачів;
— модифікування ролей користувачів;
— реєстрація у системі;
— авторизація у системі.

Інші контролери відповідають за бізнес-функції. Так, контролер DashboardDataController використовується для графіків на fronend частині і містить в собі бізнес логіку для агрегації даних. Для того, щоб отримати коректну відповідь frontend має передати у запиті ряд параметрів, таких як ідентифікатор пристрою та період часу за який проводиться агрегація. Логіка за якою формується відповідь представлена на рисунку 3.7.

Контролер RegistryController призначений для здійснення CRUD операцій над сутністю ControllerRegistry, тобто дозволяє користувачам реєструвати нові пристрої, видаляти їх, модифікувати існуючі записи та переглядати їх.

TelemetryDataController призначений для роботи з сутністю TelemetryData, він використовується для перегляду історії даних що були отримані від пристроїв. Контролер віддає дані лише для пристроїв того користувача, які були ним зареєстровані.

Секція Helpers містить спеціальні допоміжні класи що спрощують розробку. Наприклад, клас з методами розширення, клас що містить усі HTTP медіа типи клас з обгорткою стандартного логеру, тощо.

Секція ViewModels містить усі класи що є моделями DTO які передаються в якості тіла запиту на контролери.

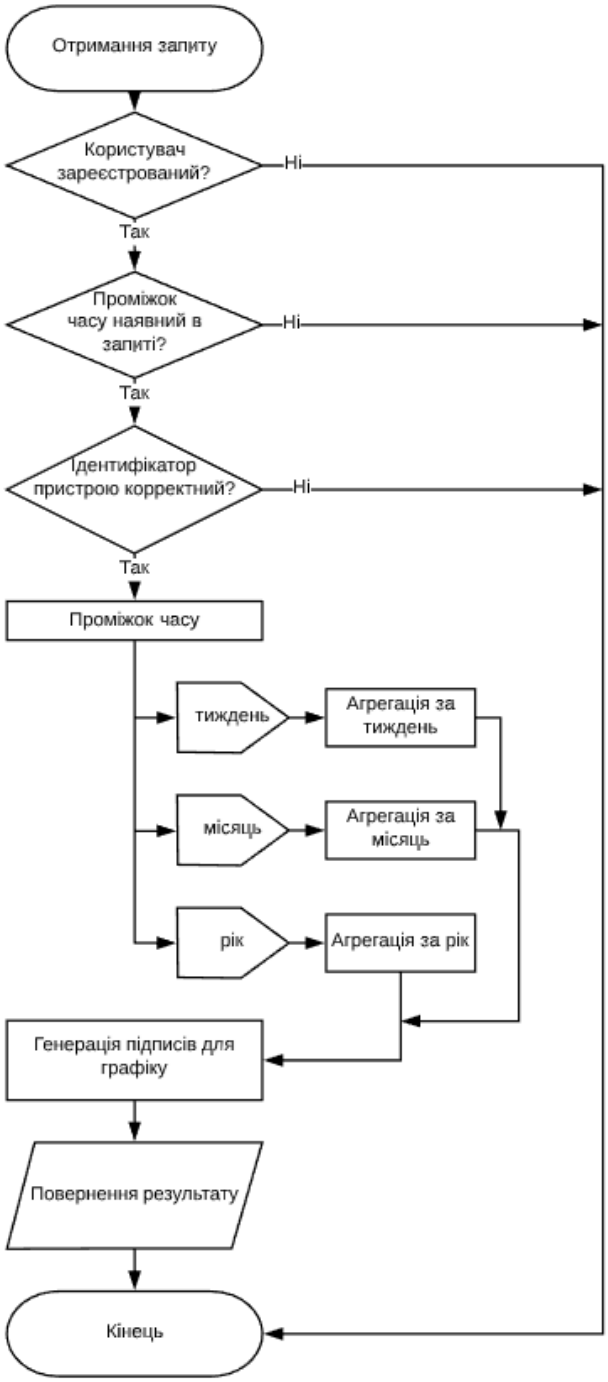

Рисунок 3.7 — Алгоритм формування агрегованої відповіді

Клас Program є вхідною точкою в додаток, в ньому конфігурується логування, реєструється клас Startup та створюється новий об’єкт IWebHost, що представляє налаштований веб-хост. Клас Startup призначений для реєстрації усіх необхідних компонентів в IOC контейнері для подальшого їх використання де це потрібно.

3.4.3 Клієнтська частина веб-застосунку

Клієнтська частина веб-застосунку була розроблена за допомогою фреймворку Angular 8 що був описаний у розділі 2. В загальному, frontend представляє з себе SPA додаток який побудовано по широковживаному шаблону MVC. Він надає графічний інтерфейс через який можна взаємодіяти з системою. Весь додаток складається з модулів, компонентів, директив і сервісів. Розглянемо структуру основних папок додатку що зображена на рисунку 3.8.

Головна сторінка знаходиться в файлі app.component, вона містить лише шапку сайту, нижню частину та блок в який роутер вставляє відповідні компоненти в залежності від поточної URL адреси. Конфігурація роутеру знаходиться в спеціальному модулі, він називається app-routing.module.

Компоненти додатку можна умовно розбити на два різних типи, компоненти що представляють собою окремі сторінки, та управляючі компоненти що мають певну частину функціоналу і підключаються до більш масштабних компонентів. Компоненти в шаблоні MVC представляють вигляд.

Моделі необхідні для типізованого представлення даних користувачу та комунікації з зовнішнім джерелом даних, backend-ом веб-застосунку.

В додатку використані лише директиви, відповідальні за наділення елемента, по відношенню до якого вони застосовуються, поведінкою, тобто атрибути.

В якості додаткового шару абстракції при роботі з джерелом даних та для приховання логіки, а отже і спрощенням роботи з JWT токеном, був створений набір сервісів.

Як було описано в першому розділі, клієнтська частина веб-застосунку в системі моніторингу має мати певний функціонал визначений вимогами, в основному це представлення даних у деякому вигляді та маніпуляція ними. При реалізації, увесь функціонал було умовно розбито відповідно до моделей даних, тобто сутностям, із якими проводяться маніпуляції.

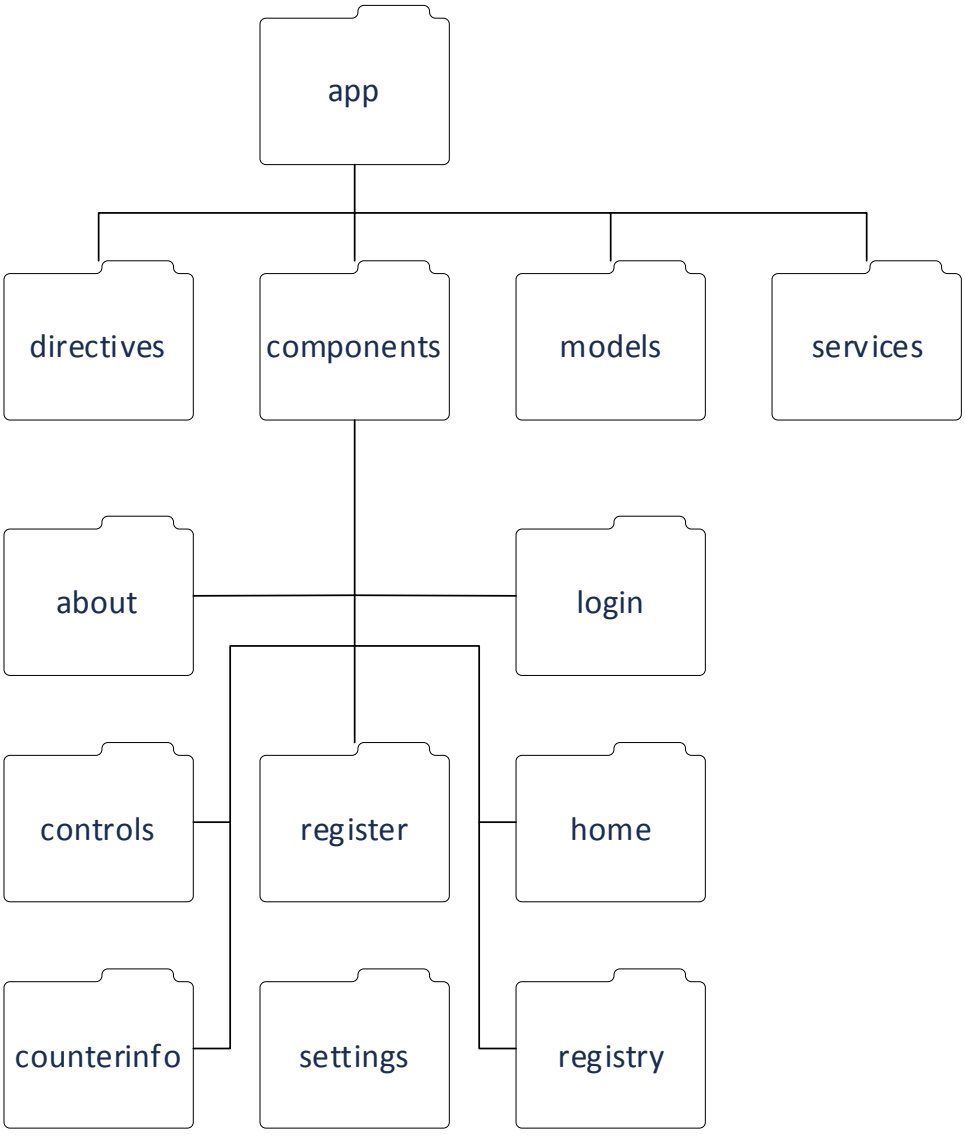

Рисунок 3.8 — Структура frontend

Для кожної сутності був створений окремий компонент, який містить у собі розмітку та необхідний функціонал для сторінки.

Такими умовними сторінками є компоненти:

— register;

— login;

— home;

— counterinfo;

— registry;

— settings.

Схема структури сторінок представлена на рисунку 3.9.

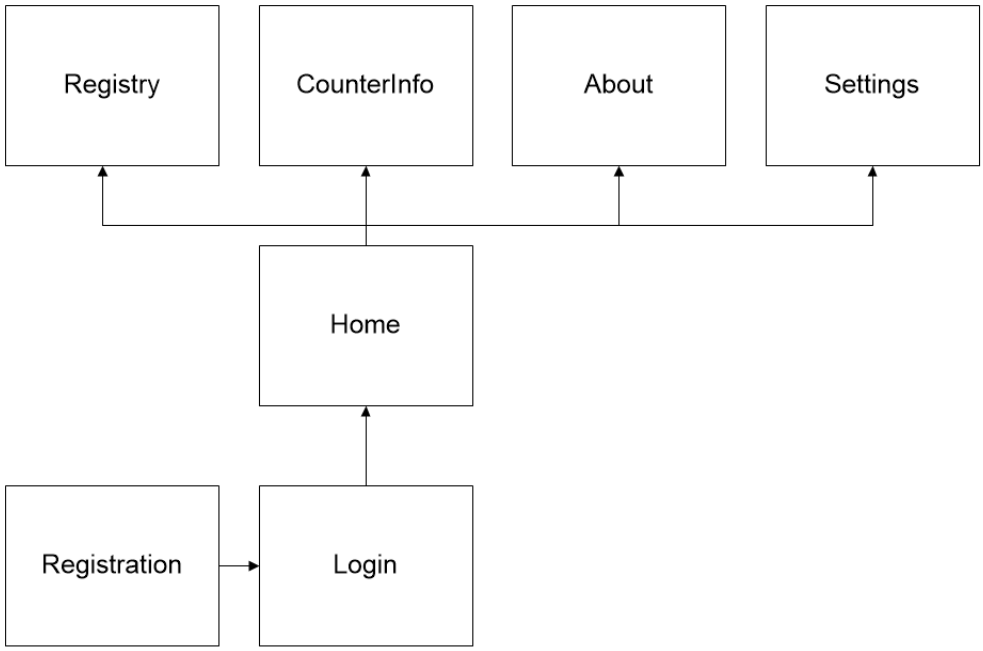

Рисунок 3.9 — Схема структури сторінок

Зі схеми очевидно, що доступ до функціональності веб-застосунку може бути отриманий користувачем тільки після проходження процесу авторизації.

Головною сторінкою є Home, в ній знаходиться функціонал по візуалізації даних у вигляді графіків та експорту побудованих візуалізацій. Графіки побудовано за допомогою бібліотеки ng2-charts.

Registry має функціонал для маніпуляцій з пристроями, їх реєстрацію, видалення та редактування уже існуючих значень.

CounterInfo призначена для перегляду історії показань, що були передані пристроями, перегляду фотографій, що були зроблені цими пристроями для подальшої валідації отриманих значень у випадку помилок чи помічених аномальних значень.

Компонент Settings складається з чотирьох менших компонентів:

— для перегляду і редактування персональних даних;

— для зміни зовнішнього вигляду додатку;

— для управління ролями;

— для управління користувачами.

Компонент About містить коротку інформацію про додаток.

3.5 Висновки до розділу

В даному розділі була розглянута програмна реалізація кожної з чотирьох підсистем в загальній системі для моніторингу. Описані методи, які використовувались при розробці, надані роз’яснення про кодову базу кожного з проєктів.

Також шляхом використання схем були описані принципи функціонування підсистем, алгоритми згідно яких вони працюють та оброблюють дані. З інформації наведеної в розділі стає зрозуміло, яким саме чином підсистеми взаємодіють одна з одною.

Незважаючи на велику кількість різних підсистем та технологій що були використані при розробці, можна сказати, що розроблена система відповідає практично усім вимогам, що були поставлені, а саме можливість подальшого масштабування системи, зручний користувацький інтерфейс для веб-застосунку, авторизація, спілкування систем між собою проходить за допомогою протоколу HTTP.

Розроблена кодова база повністю сумісна з хмарною платформою Microsoft Azure і може бути розгорнута там без великих зусиль.

4 РОЗГОРТАННЯ ТА СУПРОВІД СИСТЕМИ

4.1 Розгортання системи

Як було описано в розділі 3, система розгортається на хмарній платформі Microsoft Azure. Такий підхід дозволяє спростити розгортання та забезпечує подальше масштабування системи в майбутньому.

В термінах Microsoft Azure, будь-що доступне в платформі для користувача називається ресурсом. Ресурси об’єднуються в логічні контейнери, їх називають ресурсними групами. Групи ресурсів можуть існувати лише в рамках деякої підписки.

Для створення ресурсів за допомогою спеціального менеджера ресурсів можна використовувати декілька способів: ARM Template[17], спеціальне REST API, спеціальне SDK, або за допомогою графічного інтерфейсу на самій веб-сторінці платформи. Отже, для розгортання системи необхідно створити відповідні ресурси для кожної зі складових системи відповідно до діаграми розгортання зображеної на кресленику ІТ61.240БАК.004 Д4:

— frontend веб-застосунку для користувача;
— backend веб-застосунку для користувача;
— обробник фотографій;
— сховище статичних даних;
— реляційна база даних.

4.1.1 Вимоги до програмного та апаратного забезпечення

Для роботи по підтримці та розгортанню системи розробник повинен мати у своєму розпорядженні машину з:

— 2-х ядерний процесор з частотою 1.6 ГГЦ;
— 4 гігабайти оперативної пам’яті;
— рекомендується використовувати SSD накопичувач;

— як мінімум 1 гігабайт вільного місця, та приблизно 20 гігабайт для встановлення програмного забезпечення;

— веб-браузер Google Chrome версії 83;

— IDE Visual Studio 19;

— акаунт на платформі Microsoft Azure.

4.1.2 Реляційна база даних

Так як веб-застосунок розроблений з використанням EF Core міграцій, необхідно лише створити саму базу даних і надати додатку рядок підключення. Уся структура бази даних разом з усіма таблицями та початковими стандартними даними буде створена автоматично при першому запуску веб-застосунок.

Для створення одиничної бази даних в хмарній платформі найзручніше використати SQL Database instance. Для цього необхідно виконати такі кроки:

— вибрати створення нового ресурсу SQL Database;

— вибрати існуючу чи створити нову ресурсну групу;

— вибрати існуючий чи створити новий SQL Server;

— вибрати ім’я та початкову кількість ресурсів;

— налаштувати мережевий доступ;

— створити на сервері спеціального користувача для веб-застосунку.

4.1.3 Створення сховища статичних даних

Створення сховища статичних даних – Blob Storage, це простіша задача ніж створення бази даних. Для цього треба зауважити, що таке сховище може існувати лише в рамках спеціального Storage Account, тому спочатку треба створити і налаштувати його. В загальному, необхідно виконати такі кроки:

— вибрати створення нового ресурсу Storage Account;

— вибрати існуючу чи створити нову ресурсну групу;

— вибрати ім’я та локацію датацентру, де буде розгорнуто сховище;

— вибрати правильну георeплікацію та налаштувати мережевий доступ;

— створити в акаунті Blob Storage, та контейнер functions-pictures у ньому.

4.1.4 Безсерверне API для обробки фото

Безсерверне API побудовано за допомогою Azure Functions, тому це робить його розгортання дуже простим. Для цього необхідно лише створити новий ресурс - Function App, при чому особливу уважність в процесі створення треба приділити вибору правильного регіону та стеку, на якому написаний додаток, у випадку дипломного проєкту це .NET Core 3.1. При налаштуванні хостинг плану за яким буде працювати додаток, обов'язково треба обрати Consumption plan, адже лише в цьому випадку буде доступна автоматична масштабованість. Після цього, необхідно буде скомпілювати проєкт та опублікувати його в створений Function App.

4.1.5 Користувацький веб-застосунок

Користувацький веб-застосунок складається з двох частин, frontend і backend. Вони представляють собою два різних веб-застосунки і зазвичай розгортуються окремо. В випадку використання платформи Microsoft Azure та фреймворків Angular та ASP.NET Core є можливість суттєво спростити розгортання, адже хмарна платформа та Visual Studio візьме частину зобов'язань на себе.

Для цього необхідно виконати наступні кроки:

— вибрати створення нового ресурсу Web App;

— вибрати або створити нову ресурсну групу та регіон;

— обов'язково треба вибрати правильний стек .NET Core 3.1;

— відкрити проєкт в Visual Studio, перейти в секцію публікація;

— вибрати Azure, вибрати ресурсну групу та створений на минулих кроках Web App.

4.2 Опис інтерфейсу

4.2.1 Сторінки входу та реєстрації

При першому вході у веб-застосунок користувач потрапляє на сторінку входу у систему та бачить форму реєстрації яка зображена на рисунку 4.1.

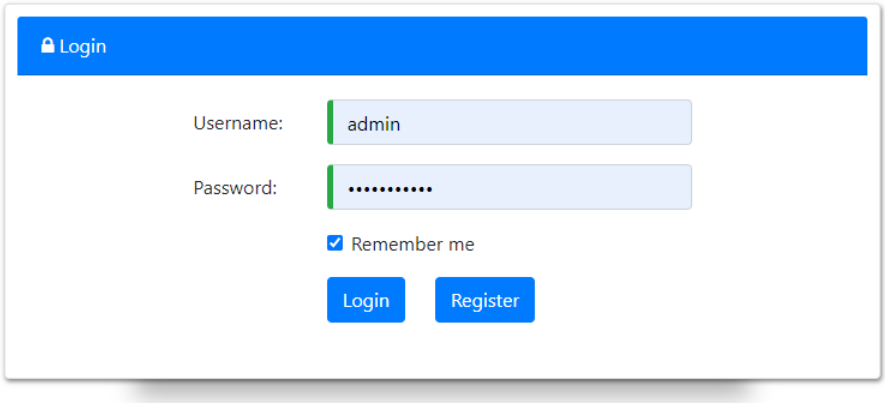

Рисунок 4.1 — Форма реєстрації

Для входу, необхідно ввести логін та пароль що були вказані під час попередньої реєстрації та натиснути кнопку Login. У випадку, коли спроба входу була неуспішною, користувач побачить відповідне спливаюче повідомлення у правому верхньому кутку (див. рисунок 4.2).

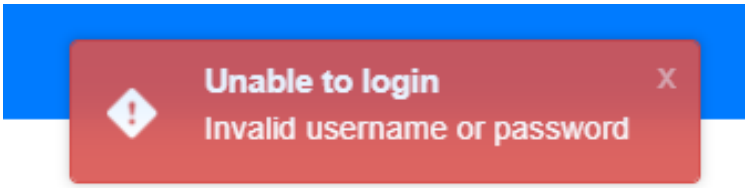

Рисунок 4.2 — Спливаюче повідомлення

В випадку коли користувач не має акаунту, йому необхідно пройти процедуру реєстрації. Для цього він переходить на сторінку реєстрації натиснувши на кнопку Register і бачить форму реєстрації (див. рисунок 4.3).

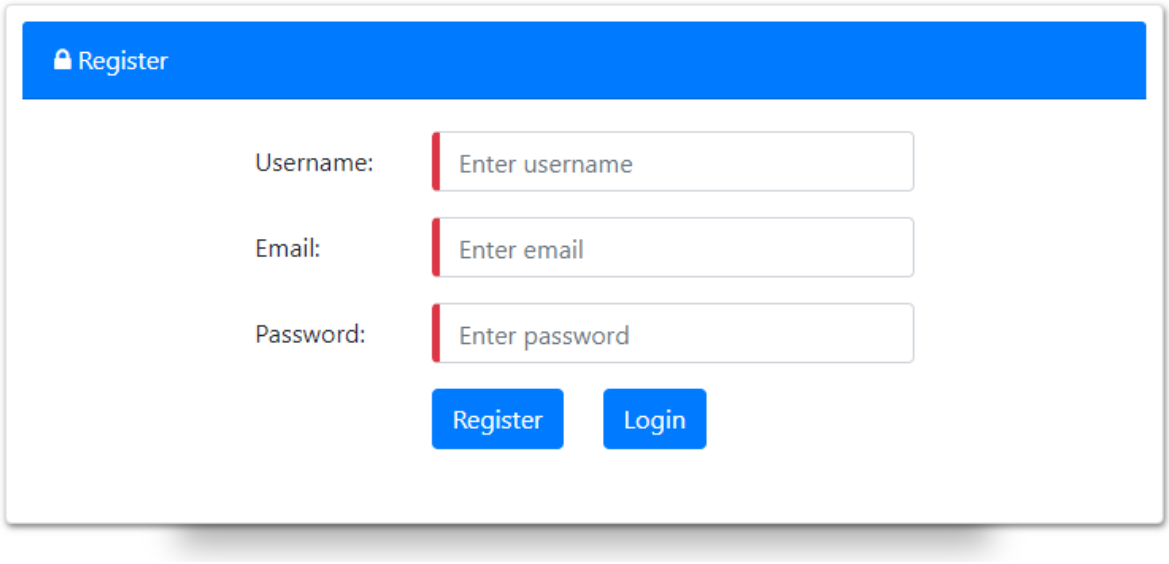

Рисунок 4.3 — Форма реєстрації

Необхідними даними для реєстрації є унікальна поштова скринька, унікальне ім’я користувача та пароль. При введені, система автоматично перевірить дані у формі на коректність, у випадку помилки з’явиться спливаюче повідомлення у правому верхньому куті з роз’ясненням причини.

Якщо реєстрація була успішною, користувач буде перенаправлений на сторінку входу, для входу у систему. Якщо вхід успішний, то користувач потрапляє на головну сторінку веб-застосунку.

4.2.2 Головна сторінка

Головна сторінка є основною робочою зоною для користувача, адже вона містить функціональність для візуалізації даних у вигляді графіків та для подальшого експорту цих графіків у форматі pdf. Скріншот сторінки представлено на рисунку 4.4. На скріншоті видно, що основну робочу область на сторінці займає

саме зображення поточного графіку, що дозволяє проводити аналіз отриманих даних максимально зручно для користувача.

Рисунок 4.4 — Головна сторінка

Користувач має можливість для кастомізації графіків за допомогою спеціальних кнопок що відповідають за різні аспекти графіку

Зліва направо це кнопки:

— експорт;

— оновлення даних;

— вибір типу графіку;

— вибір проміжку часу;

Кнопка експорт конвертує зображення графіку у pdf документ і автоматично починає завантаження цього документу на клієнтську машину. Кнопка оновлення даних відповідає за отримання нових даних з серверу.

Кнопка вибору графіку є селектором з випадаючим списком типів графіків, наявні такі варіанти:

— лінійний;

— стовпчаста діаграма;

— кругова діаграма;

— полярна діаграма;

— радарна діаграма;

— кільцева діаграма.

Кнопка вибору проміжку часи, є селектором з випадаючим списком можливих проміжків часу для агрегації і подальшого відображення, наприклад на рисунку 4.5 відображені дані за минулий місяць. Пропонуються варіанти агрегації за тиждень, місяць та рік. Агрегація проводиться шляхом обчислення середнього значення лічильника за день або за місяць в залежності від вибраного проміжку.

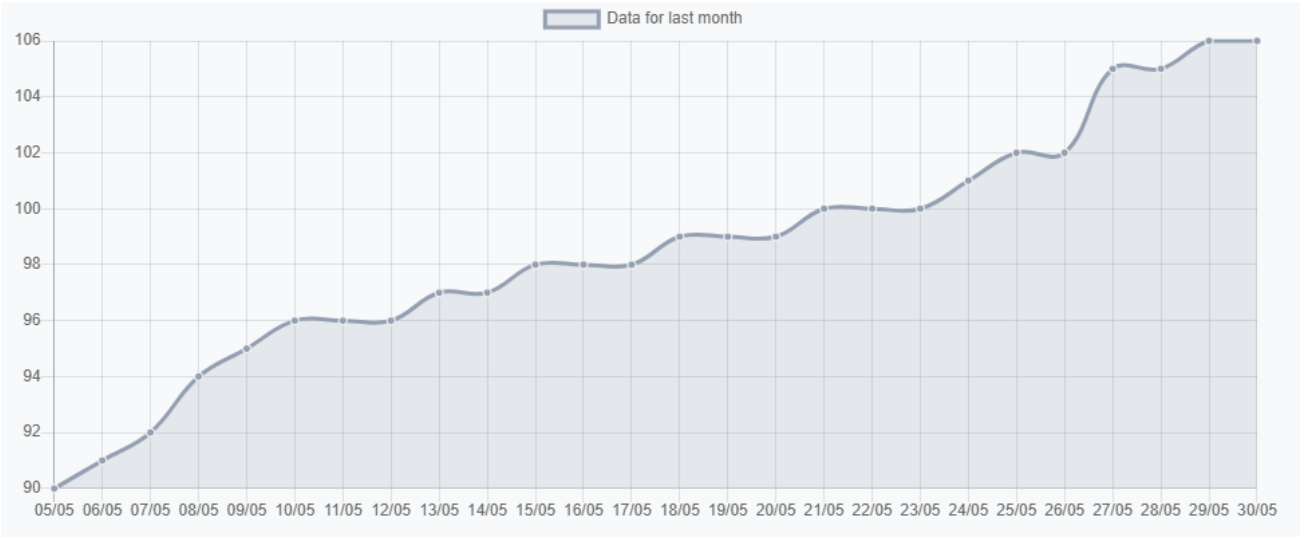

Рисунок 4.5 — Приклад відображення даних за місяць

4.2.3 Реєстрація нового пристрою обліку

Передбачується, що користувач має мати змогу для моніторингу багатьох лічильників. Для цього призначена сторінка Registry. Процес реєстрації нового пристрою, редактування та перегляд інформації про вже зареєстровані пристрої доволі простий і проводиться на сторінці Registry (див. рисунок 4.6). Для редактування імені чи опису необхідно двічі клацнути

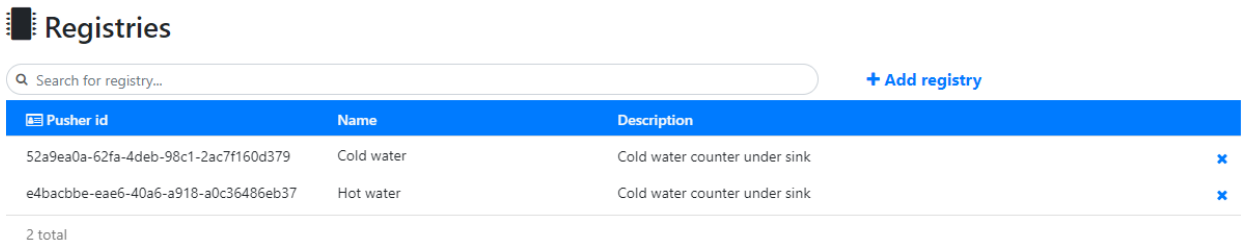

Рисунок 4.6 — Сторінка Registry

Для реєстрації нового пристрою необхідно натиснути на кнопку Add registry. Відкриється форма реєстрації (див. рисунок 4.7).

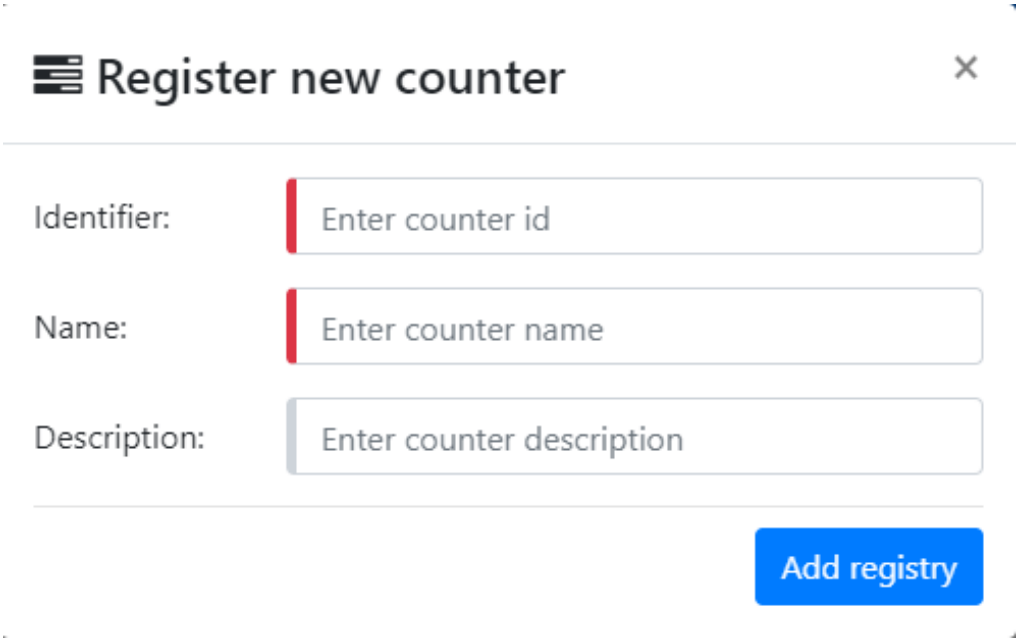

Рисунок 4.7 — Форма реєстрації нового пристрою

Необхідною інформацією для процедури реєстрації є унікальний ідентифікатор пристрою у форматі GUID та ім’я для нового пристрою. У випадку помилки при створенні запису вона буде відображена у верхньому правому куті у вигляді спливаючого повідомлення.

4.2.4 Перегляд історії значень засобів обліку

Сторінка історії значень (див. рисунок 4.8) призначена для перегляду історії записів та валідації отриманих системою значень. На сторінці міститься

таблиця з усіма коли-небудь отриманими значеннями, таблиця може бути відсортована за будь-якою колонкою, для цього необхідно клікнути на назву стовпця. Також, для полегшення орієнтування в таблиці, пропонується використовувати пошуковий рядок який шукає значення у всіх колонках.

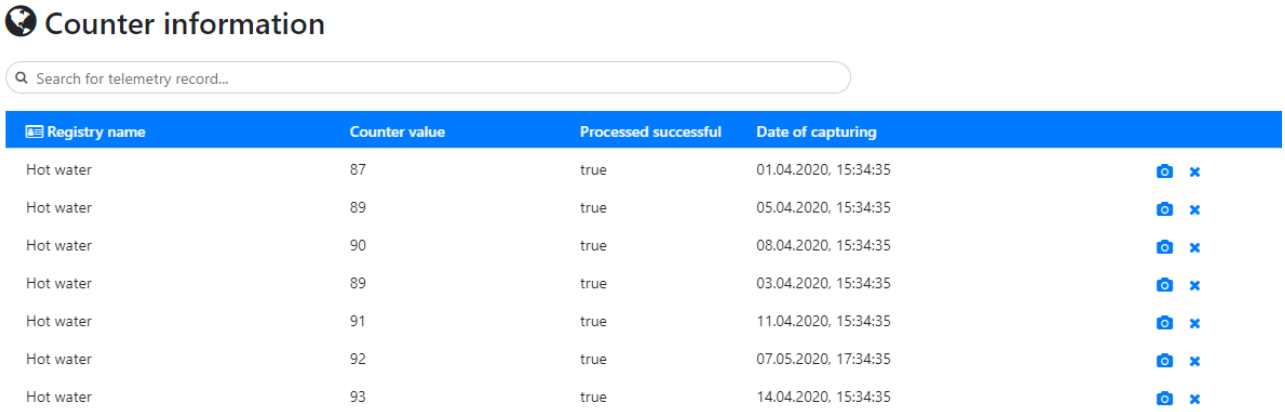

Рисунок 4.8 — Сторінка історії значень

У кожному рядку таблиці, справа міститься дві кнопки. Зліва направо: кнопка перегляду фото та кнопка видалення запису. Кнопка перегляду фото необхідна для кінцевої валідації отриманого користувачем значення у випадку якщо ним була помічена деяка аномалія, наприклад дуже високий скачок рівня споживання. Для редактування значення лічильника, необхідно клікнути два рази на бажане для редакції поле і ввести нове значення, після чого воно автоматично збережеться.

4.2.4 Профіль користувача

Сторінка з профілем користувача (див. рисунок 4.9) необхідна для перегляду та заповнення користувачем особистої інформації, наприклад адреси поштової скриньки чи номера мобільного телефону для зв’язку з ним. Ця інформація необхідна для персоніфікації пропозицій для користувача та зв’язку з ним при необхідності.

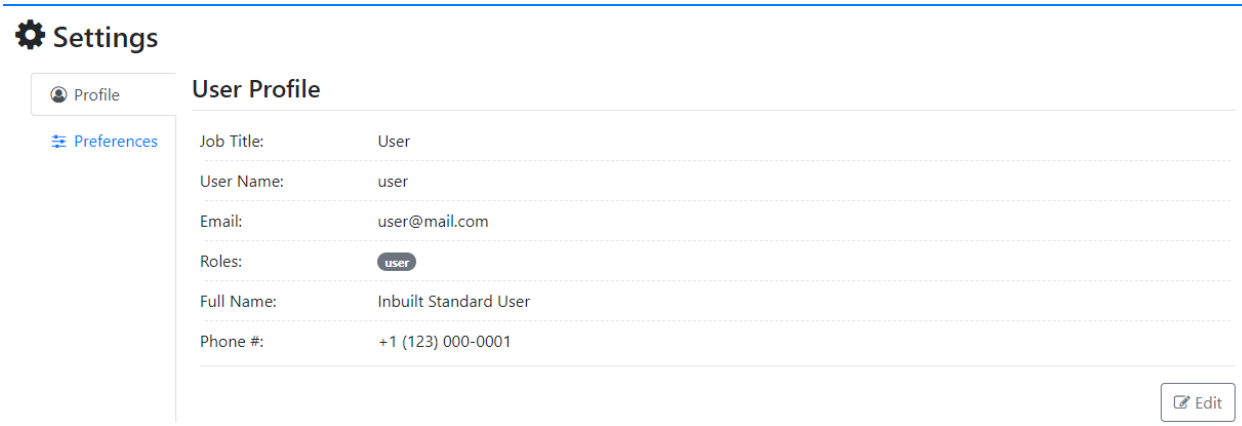

Рисунок 4.9 — Профіль користувача

Для переходу в режим редагування необхідно натиснути на кнопку Edit, на місці статичних написів з'явиться форма (див. рисунок 4.10). В поля необхідно ввести нові дані та натиснути кнопку Save. При виникненні будь-яких помилок валідації чи при збереженні нових даних, користувач побачить повідомлення про це у верхньому правому куті у вигляді спливаючого повідомлення.

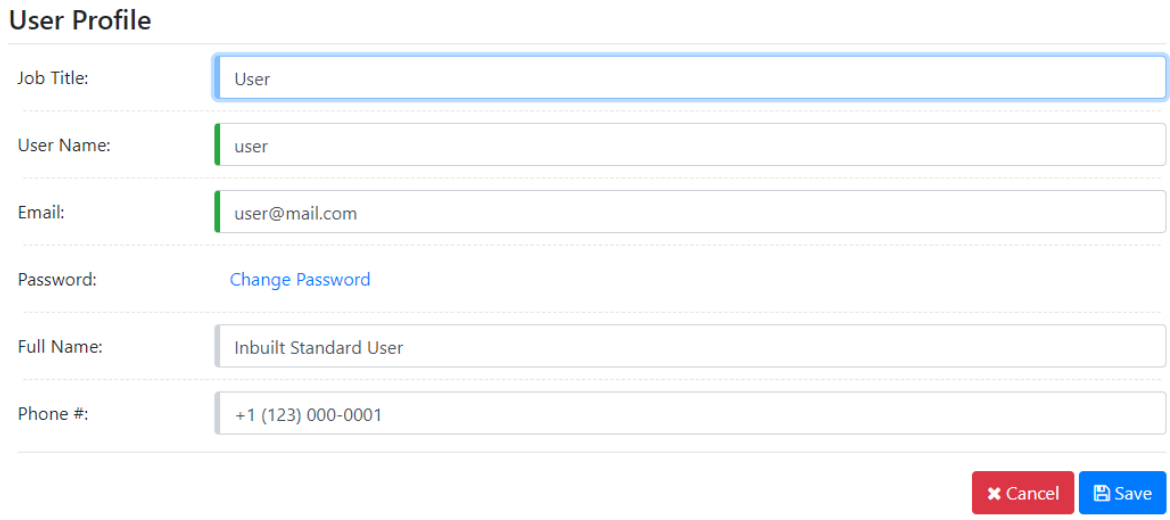

Рисунок 4.10 — Форма редактування профілю

4.2.5 Налаштування зовнішнього вигляду

Сторінка налаштувань зовнішнього вигляду веб-застосунку (див. рисунок 4.11) має декілька опцій для зміни користувачем: мову та тему. У випадку якщо

користувач вирішить скинути ці налаштування до стандартних, має бути натиснута кнопка Reload preferences яка і здійснить цю операцію.

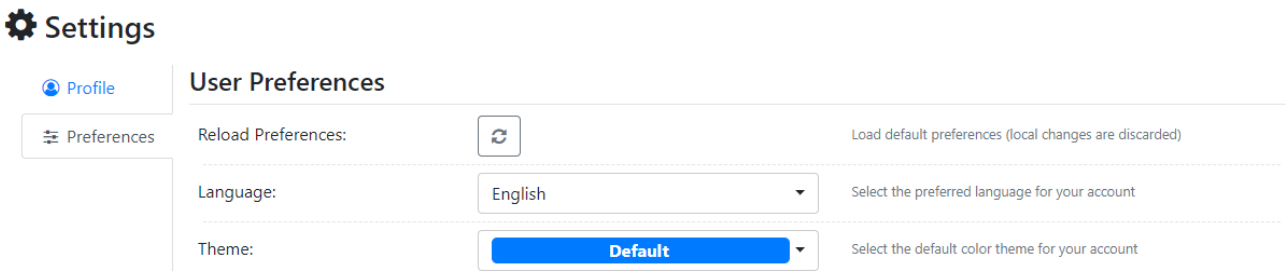

Рисунок 4.11 — Сторінка налаштувань зовнішнього вигляду

4.2.6 Управління ролями та користувачами

У випадку якщо поточний користувач є адміністратором, йому доступні додаткові опції такі як управління користувачами та ролями. Управління користувачами здійснюється на спеціальній сторінці з однойменною назвою (див. рисунок 4.12).

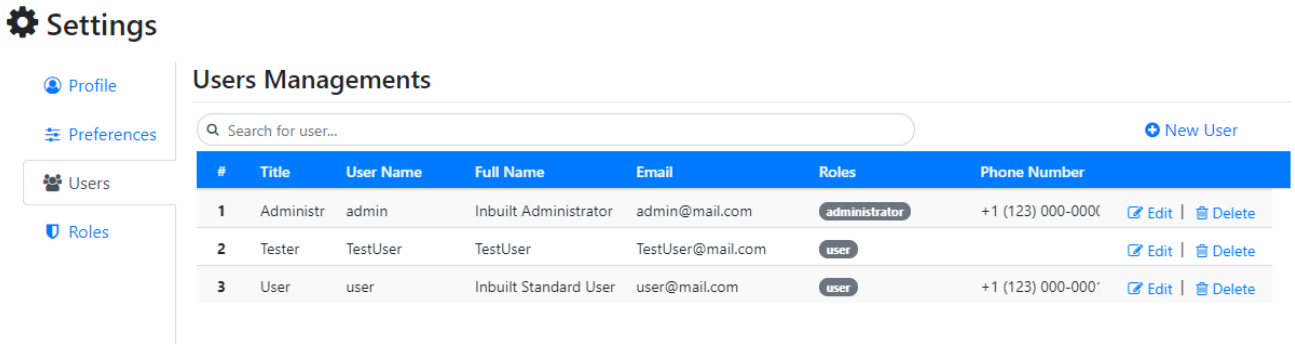

Рисунок 4.12 – Сторінка управління користувачами

Користувацький інтерфейс цієї сторінки в основному займає таблиця зі списком користувачів. Кожен рядок містить кнопки видалення та редагування користувача. При натисканні на кнопку редагування відкриється форма редагування користувача (див. рисунок 4.13). Дана форма дає можливість редактувати

усю можливу інформацію про користувача включаючи надання прав шляхом зміни ролі та відключення акаунту.

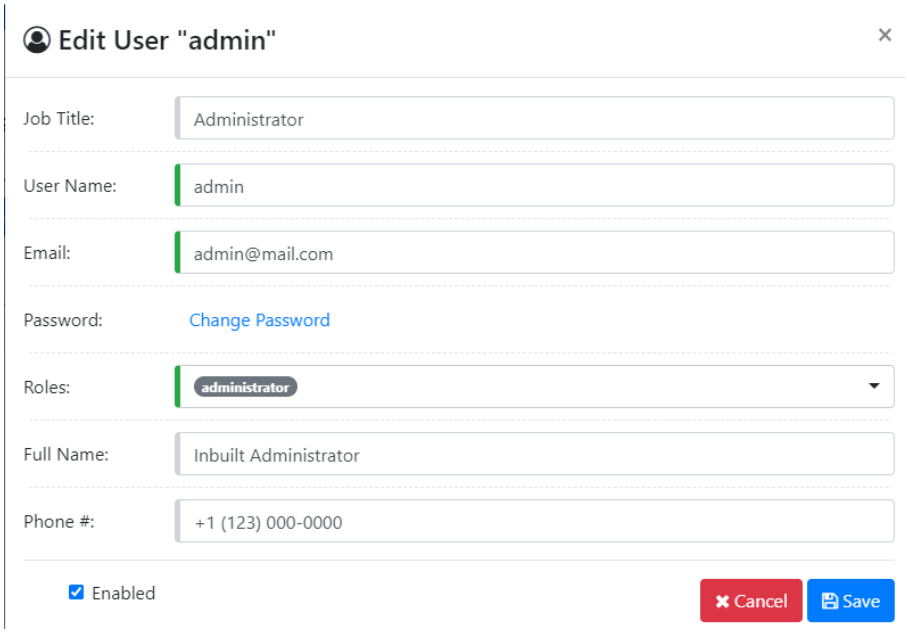

Рисунок 4.13 – Форма редагування користувача

Для управління ролями у системі призначена сторінка управління ролями (див. рисунок 4.14). Як і сторінка управління користувачами, вона містить основну таблицю з списком усіх існуючих ролей.

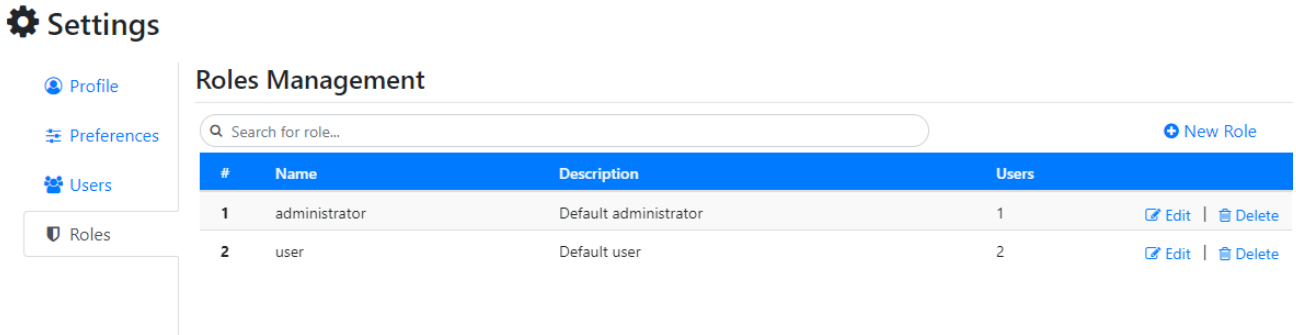

Рисунок 4.14 – Сторінка управління ролями

Для перегляду деталей конкретної ролі чи редактування прав які видані, необхідно натиснути на кнопку. Після цього, відкриється відповідна форма в якій і можна провести усі операції.

4.3 Висновки до розділу

В даному розділі було описано, яким чином відбувається розгортка кожної з підсистем та супутніх ресурсів на хмарній платформі Microsoft Azure в покроковій формі. Сам процес розгортки був описаний використовуючи один із способів розгортки ресурсів, що надається платформою, а саме інтерфейс веб-порталу.

Також, розділ містить вимоги то технічного та програмного забезпечення розробника, що буде виконувати задачі по розгортуванню та подальшій підтримці системи.

Було приділено особливу увагу детальному опису графічного інтерфейсу веб-застосунку з яким має взаємодіяти користувач, адже додаток складається с доволі великої кількості веб-сторінок для управління пристроями для моніторингу, для візуалізації даних та експорту даних, перегляду історії отриманих від пристроїв обліку значень, управління персональною інформацією, адміністрування, тощо.

ВИСНОВКИ

В рамках роботи над дипломним проєктом був реалізований програмно-апаратний комплекс, що являє собою систему для моніторингу засобів обліку. Така система дозволяє автоматизувати та полегшити рутинну задачу зі зняття даних з засобів обліку та подальшої роботи з цими даними. Розроблена система може бути розгорнута на хмарній платформі або локально.

При розробленні даної системи роботу було розбито на декілька етапів. Спочатку було проаналізовано існуючі технічні рішення для цієї задачі, потім на основі цієї інформації було розроблено власні функціональні і нефункціональні вимоги до системи.

Після того як були сформульовані вимоги до системи моніторингу, було обґрунтовано вибір ряду технологій та підходів для виконання задачі по розробці системи. Обрані технології дозволили розробити систему таким чином, що її функціонал легко може бути розширений, а сама система має можливість до масштабування у подальшому.

Також, за результатами розробки була описана програмна реалізація проєкту, наведений набір діаграм для опису побудованої системи у графічному вигляду, описано графічний інтерфейс користувача та наведений опис по розгортуванню системи на хмарній платформі.

Загалом, побудована система є сучасною, відповідає поставленим на початку роботи вимогам як в рамках функціоналу так і в технічному аспекті а, веб-застосунок для взаємодії з користувачем має зручний користувацький інтерфейс.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Quality Assurance URL: http://lvivqaclub.blogspot.com/2008/10/blog-post_17.html (Last accessed 05.04.2020).
2. Role-based access control URL: https://searchsecurity.techtarget.com/definition/role-based-access-control-RBAC (Last accessed 06.04.2020).
3. Специфікація протоколу HTTP URL: https://tools.ietf.org/html/rfc2616 (Дата звернення 07.04.2020).
4. Порівняння типів архітектури систем сервісів / Петренко А. А. / Системні дослідження та інформаційні технології, 2015, № 4. – ст. 51
5. SOLID Principles made easy URL: https://medium.com/@dhkelmendi/solid-principles-made-easy-67b1246bcdf (Last accessed 08.04.2020).
6. .NET Core vs .NET Framework for server apps URL: https://docs.microsoft.com/en-us/dotnet/standard/choosing-core-framework-server (Last accessed 08.04.2020).
7. Docker overview URL: https://docs.docker.com/get-started/overview/ (Last accessed 08.04.2020).
8. Introduction to NuGet URL: https://docs.microsoft.com/en-us/nuget/what-is-nuget (Last accessed 12.04.2020).
9. JSON Web Token (JWT) URL: https://tools.ietf.org/html/rfc7519 (Last accessed 12.04.2020).
10. Repository pattern URL: https://martinfowler.com/eaaCatalog/repository.html (Last accessed 13.04.2020).
11. What is IaaS? URL: https://azure.microsoft.com/en-us/overview/what-is-iaas/ (Last accessed 14.04.2020).
12. What is PaaS? URL: https://azure.microsoft.com/en-us/overview/what-is-paas/ (Last accessed 14.04.2020).

13. SPA vs MPA URL: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58 (Last accessed 15.04.2020).

14. ESP32 Overview URL: https://www.espressif.com/en/products/socs/esp32/overview (Last accessed 15.04.2020).

15. Introduction to Azure Blob storage URL: https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction (Last accessed 16.04.2020).

16. What is REST URL: https://restfulapi.net/ (Last accessed 16.04.2020).

17. Azure resource manager overview URL: https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/overview (Last accessed 16.04.2020).